VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY
A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

NÁVRH VIRTUÁLNÍHO SÍŤOVÉHO KOLABORATIVNÍHO ZVUKOVÉHO NÁSTROJE

DESIGN OF NET-BASED VIRTUAL COLLABORATIVE MUSICAL INSTRUMENT

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. Denis Liudkevich
AUTHOR

VEDOUCÍ PRÁCE — MgA. Jan Kavan, Ph.D.
SUPERVISOR

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní obor **Audio inženýrství**

Ústav telekomunikací

Student: Bc. Denis Liudkevich ***ID:*** 164890 ***Ročník:*** 2 ***Akademický rok:*** 2019/20

NÁZEV TÉMATU:

Návrh virtuálního síťového kolaborativního zvukového nástroje

POKYNY PRO VYPRACOVÁNÍ:

Prozkoumejte existující online kolaborativní virtuální hudební nástroje a realizujte vlastní nástroj s inovativními prvky. V rámci semestrálního projektu vyberte vhodné vývojové a provozní prostředí a tento výběr zdůvodněte, proveďte analýzu případů užití nástroje a procesní analýzu, navrhněte wireframe uživatelského rozhraní, infrastrukturu s ohledem na škálovatelnost a síťový komunikační protokol. Dále specifikujte zamýšlené zvukové moduly a vytvořte jednoduchý nástroj pro základní ověření konceptu. Popište také rizika a omezení plánovaného řešení. V rámci navazující diplomové práce nástroj realizujte a otestujte v reálném provozu.

DOPORUČENÁ LITERATURA:

[1] WILSON, Scott, Nick COLLINS a David COTTLE. The SuperCollider book. Cambridge, Mass.: MIT Press,2011. ISBN 978-0262232692.

[2] FILIMOWICZ, Michael. Foundations in sound design for interactive media: a multidisciplinary approach. NewYork, NY: Routledge, 2019. ISBN 9781138093942.

Termín zadání: 3.2.2020 ***Termín odevzdání:*** 1.6.2020

Vedoucí práce: MgA. Jan Kavan, Ph.D.

rof. Ing. Jiří Mišurec, CSc.
ředseda oborové rady

UPOZORNĚNÍ:

Autor diplomové práce nesmí při vytváření diplomové práce porušit autorská práva třetích osob, zejména nesmí zasahovat nedovoleným způsobem do cizích autorských práv osobnostních a musí si být plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č.40/2009 Sb.

Abstrakt:

Cílem této práce bylo vytvořit online platformu pro zvukovou tvorbu, určenou pro více uživatelů, s originálními nástroji syntézy zvuku. Byl zohledněn i edukativní kontext aplikace uschováním ovladačů parametrů zvuku za podvědomě známé fyzikální jevy a herní podobu aplikace. Podstatná část logiky a veškerá grafika nástrojů je napsána v programovém jazyce JavaScript a jeho knihovně p5.js. Nachází se na straně klienta a komunikuje se serverem na bázi Node.js pomocí websocketu. Zvuková část je na zvláštním serveru v prostředí SuperCollider, vysílá se pomocí IceCastu a komunikuje s hlavním serverem OSC zprávy. Aplikace obsahuje 3 nástroje ke generování zvuků a jeden efektový modul. Každý z nástrojů je určen pro více uživatelů a vyžaduje jejich spolupráci. Optimizací interních algoritmů nástrojů, volbou způsobu zobrazování grafických obsahů a správným propojením jednotlivých zvukových modulů bylo dosaženo přijatelných přenosových rychlostí a minimálních výpočetních nároků. Zvuk je charakteristický pro každý nástroj, nástroje v aplikaci jsou odladěné a navržené tak, aby uživatel mohl jak dosáhnout zajímavých zvukových výsledků sám, tak i zahrát svoji roli v celku s ostatními. Ke generaci zvuku jsou použity takové metody jako granulární syntéza, chaotické oscillátory, modelování strunných nástrojů, kombinace filtrů a tak dále. Velký důraz při vývoji aplikace byl kladen na rozdělení rolí, společné ovládání jednoho nástroje více hráči a komunikaci uživatelů prostřednictvím hry na nástroje a slovní projev - chat. Nedílnou součástí je taky blok pro zobrazování popisující informace.

Abstract:

The aim of this work was to create an online platform for multi-user sound creation with original sound synthesis tools. The educational context of the application was also taken into account by hiding the controls of the sound parameters behind the subconsciously known physical phenomena and the game form of the application. A substantial part of the logic and all graphics of the instruments is written in the JavaScript programming language and its library p5.js. It is located on the client side and communicates with the Node.js-based server via a web socket. The audio part is on another server in the SuperCollider environment, it is transmitted via IceCast and communicates with the main OSC message server. The application contains 3 instruments for generating sounds and one effects module. Each instrument is designed for multiple users and requires their cooperation. Acceptable transmission speeds and minimum computational demands have been achieved by optimizing the instrument's internal algorithms, the way in which the graphic content is displayed and the appropriate routing of the individual sound modules. The sound is specific for each instrument. The instruments in the application are tuned and designed so that the user can both achieve interesting sound results himself and play his role as a whole with others. Methods such as granular synthesis, chaotic oscillators, string instrument modeling, filter combinations, and so on are used to generate sound. Great emphasis in the development of the application was placed on the separation of roles, simultaneous control of one instrument by several players and communication of users through playing the instruments and text expression - chat. An important part is also a block for displaying descriptive information.

Klíčová slova:

Online, kolaborativní, zvukový, herní, design, audio, syntéza.

Keywords:

Online, collaborative, sound, game, design, audio, synthesis.

Bibliografická citace:

LIUDKEVICH, Denis. Návrh virtuálního síťového kolaborativního zvukového nástroje. Brno, 2020. Dostupné také z: https://www.vutbr.cz/studenti/zav-prace/detail/126061. Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce Jan Kavan.

Prohlašuji, že svou diplomovou práci na téma „Návrh virtuálního síťového kolaborativního zvukového nástroje“ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce. Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

V Brně dne . . . . . podpis autora . . . . . .

Děkuji vedoucímu diplomové práce MgA. Janu Kavanu, Ph.D., za cenné rady, nekonečnou inspiraci a podporu při zpracování práce.

V Brně dne . . . . . podpis autora . . . . . .

Obsah

SEZNAM OBRÁZKŮ

1. ÚVOD

Na základě rešerše spektralistické experimentální hudby, hudby tembrální a průzkumu aktuálního trhu nástrojů, vytvořených za účelem kolaborativní tvorby zvukového materiálu, je evidentní, že je potřebné začít vyvíjet nové víceuživatelské online hudební nástroje ve formě webové aplikace.

Po počátečním uvedení do dané problematiky zkoumám ve druhé kapitole principy interaktivity a možnosti designu rozhraní jak webové aplikace, tak zvukového nástroje. Přihlížím k možným problémům začínajících uživatelů a nevyhnutelnému faktu, že se budou při tvorbě nacházet v jednom prostředí společně s dalšími, více zkušenými uživateli. Jsou zkoumány výhody počítačové hudby a číslicové syntézy. Předkládám také psychologický pohled na přístup a spolupráci při zvukové tvorbě. Z existujících projektů, věnujících se tomuto tématu, byly vybrány a popsány ty, které se shodovaly buď použitými technologiemi nebo záměrem s mou prací.

Později v práci následuje samotný návrh jednotlivých nástrojů a jejich aplikace. Ty jednak doplňují chybějící možnosti vzorových projektů a zároveň kombinují jejich silné stránky. Kapitola obsahuje podrobné schéma struktury grafického rozhraní a blokové schéma systémového modelu.

Následně se v práci uvádí souhrn a zdůvodnění použitých technologických postupů.

Před kapitolou o realizaci nástroje je popsán teoretický základ zvukové syntézy, vybraných metod pro modulaci a základní postupy zpracování zvukových signálů.

Další kapitola je věnována realizaci programu, jeho serverové a klientské struktury a na příkladu kódu je popsána vzájemná komunikace těchto částí programu. Jsou zde řešené i grafické prvky ovládání a zobrazovací panel. Také je popsána provázanost generování zvuků na grafiku.

Ke konci jsou předloženy metody pro optimalizaci na úrovni programování zvukových modulů, metody pro minimalizaci rizik pramenících z uživatelského chování, které by mohlo narušit hladký chod aplikace, např. útok hackera.

2. HISTORICKÝ HUDEBNÍ ZÁKLAD

V této kapitole představím problematiku experimentálních elektronických hudebních nástrojů i motivaci a potřeby hudebníků, kteří budou užívat nástroje ke své tvorbě. Představíme je na příkladu dvou hudebních směrů, které položily základy určitému myšlení, jenž je klíčové v rámci tohoto projektu. Skladatelé těchto směrů rozvinuli experimentální hudbu a posunuli chápání hudby obecně - poukázali na možnost hudebního použití neharmonických zvuků a elektronických hudebních nástrojů. Převedli pozornost od tonality k faktuře zvuku, jeho barvě a spektrálnímu složení.

2.1 Tembrální hudba

Tembrální hudba je hudební termín, popisující přístup k tvorbě skladatelů určitého období polské školy. Založil jej teoretik Józef M. Chomiński a jeho původní definice popisuje tembrální hudbu následovně: "focus on purely sonorous values as the main means of expression, and thus a structural element of a composition". [1] Jinými slovy cílem tohoto hudebního směru je hledání nového zvuku, témbru, barvy a textury a tento cíl je upřednostňován před výškou tónu a před vztahy mezi tóny samotnými. Proto například nebylo neobvyklé v rámci tohoto odvětví hudby použití non-functional akordů za účelem tembrálního efektu. [1] K dosažení těchto zatím neobjevených zvukových barev se používaly méně rozšířené nástroje, skladatelé se nebáli hledat nástroje nové a užívat je v různých kombinacích společně s nástroji tradičními. Dokonce i z klasických nástrojů se dalo těžit nestandardními technikami hry. Svého vrcholu toto hnutí dosáhlo v avantgardní hudbě 1950-60 let, kde se už počítalo s kompletně jinými texturami a hudebními zákony. [2] V době působení tembralistů si jejich inovací samozřejmě všímali i v zahraničí a například Danuta Mirka se o nich vyjadřoval takto: “The unifying features of this music were sought chiefly on the aesthetic plane, in its strong, ardent expression and the dynamism of its formal processes. Both of these qualities were

equally strange to the experimental ‘asceticism’ of Western music in the 1950s and hence were all the more noticeable in the music coming from Poland.” [3]

Francouzští skladatelé té doby se také poněkud odchýlili od soustředění se na funkční koncepty a směřovali spíše k hledání a pochopení čistého tónu. Tento jev je výrazný ve tvorbě koloristy Debussyho. Podle Chominského by se ale podobné myšlenky daly dohledat již v roce 1913 u vídeňských skladatelů nebo u A. Weberna například ve skladbě Die Sonne, kde není snadné jednoduše sledovat melodii nebo harmonii, jelikož funkci akordů převzala spíš oblast tembrální, rytmická a kontrastní. Zdrojem inspirace a nových zvuků se v tuto dobu staly nové elektronické hudební nástroje, které svou existencí a funkcí znázorňovaly samotnou esenci myšlenek tembralistů, hlavně v možnostech pracovat se spektrem a strukturou zvukové vlny.

U tembrální hudby je obtížné hovořit o konkrétních výrazových prvcích. Charakter a přístupy tembrální hudby podle Chominského mohou být popsány jenom obecně. Proto můžeme popisovat jenom pojmy vertikálních a horizontálních struktur, ve kterých se nacházejí všechny zvukové objekty, jako jsou například klastry, které nahradily klasické akordy, melodie a noty. [4] Tembrální hudba rozhodně ovlivnila skladatele své doby a samozřejmě i mnoho dalších až do současnosti. Principy tvorby této školy byly revoluční a otevřely novou perspektivu a nový pohled jak na analýzu starších děl, tak i na díla budoucí.

Hlavně však měly podstatný vliv na směr vývoje současných hudebních nástrojů i na vývoj současné hudby jako takové.

2.2 Spektralismus

Zvuk jako fyzický fenomén dominoval v představách a fantaziích avantgardních skladatelů konce 20. století. Po 60. letech tito skladatelé začali odmítat myšlenky serialistů a komplexnost matematického popisu a tím vznikl prostor pro dvě nové vlny: minimalismus v Americe a spektralismus v Evropě. Xenakis je jedním z prvních skladatelů, který začal hledat hlubší strukturu v harmonickém spektru. Jeho následovníky se v 70. letech stali G. Grisey, T. Murail a

H. Radulescu. Jejich hudbu je možné popsat jako šumovou, přízračnou a nadpřirozenou. Jak pravil Jonathan Harvey: ”Hodně spektralistů používalo zkreslené tóny nebo šumy, tvořené komplexním harmonickým spektrem. Často se v jejich tvorbě setkáváme s perkusními nástroji a užitím každodenních zvuků. Už neexistuje tak ostrá hranice mezi šumem a hudebním tónem.” Vize těchto tvůrců odsuzuje klasické vnímání hudby jako posloupnost not a jejich kombinaci v čase, zvuk je vnímán jako živá instance a tak se s ní také zachází. Sami tvůrci však nepovažují spektralismus za hudební směr, ale vnímají ho jako techniku skladby (Radulescu) nebo jako pohled, vztah ke skladbě (Grisey).

Předchůdce spektrální hudby Xenakis, když pracoval s mikrotonalitou a zvukovými masami, měl podobnou úvahu a představu o hudbě. Odmítal rozdělovat stupnici na základních 12 půltónů, stejně tak odmítal tonální strukturu jako takovou, zároveň ale na rozdíl od spektralistů nekladl velký důraz na bohaté spektrum.

Ačkoli první skladatelé spektrální hudby ve velké míře nepoužívali elektronické hudební nástroje, byli jimi určitě inspirováni nebo alespoň ovlivněni. Nebyli to technici, ale snažili se o technickou analýzu zvuku, o preciznost a přesnost. Používání elektronických nástrojů se vyhýbali hlavně z důvodu možného opotřebení a zastaralosti nástrojů v budoucnu, a proto stále preferovali klasické orchestrální techniky.[5]

Současná doba po nás však vyžaduje hledání “nového zvuku” už v první fázi návrhu nástrojů, ať už elektronických hardwarových nebo softwarových a je i mým cílem najít další možnou cestu tvorby zvuku nebo i sebemenší novou a netradiční barvu zvuku.

3. INTERAKCE, INTERAKTIVITA A INTERFACE

Zvuk můžeme pokládat za prostředek ke komunikaci, který může nabývat mnoha významů, například: uchovávání informace pomocí záznamů, vztahy tónů a barvy zvuku, které nesou emoce, sémantický význam slov a řeči, psychoakustické a fyziologické signály typu upozornění, podvědomé vjemové působení, předávání energie zvukovou vlnou atp. Každý z těchto významů si zaslouží pozornost a zvláštní zamyšlení při návrhu nového nástroje, ovšem každé jednotlivé téma je natolik obsáhlé, že se každému věnuje samostatná věda. Proto se v rámci této kapitoly zaměříme hlavně na uživatelské prostředí a interakci mezi člověkem a počítačovým programem.

Proč vůbec počítačová hudba? Důvodů je více:

- Rychlost. Při zrození nové myšlenky nikdy není jasné, jestli za něco stojí a možnost rychle zkoušet a testovat nápady je při experimentální tvorbě rozhodující. Se skutečným orchestrem nebo alespoň s kapelou musíte organizovat hodně věcí: najít lidi, domluvit si termín, vhodnou místnost, donutit každého naučit se svůj part. Nastudovat software a také práci s počítačem sice zabere čas, ale jakmile si tímto procesem jednou projdete, získáte možnost realizovat nápady a myšlenky během minut, zatímco s tužkou a hráči potřebujete stavět pokaždé od začátku.
- Přesnost. Počítač je schopný zahrát časové nebo tónové rozdíly s přesností na tisíciny. Pro člověka to možné není. Napodobit tento lidský faktor v počítači je do určité míry možné pomocí náhodných procesů a algoritmu odchylek, ale natrénovat počítačovou přesnost hráči možné není. Nenaznačuji, že veškerá hudba musí být interpretována počítačem, ale toto poznání nám může prospět v dalších nápadech, notaci a edukačních procesech.

- Složitost. Lidský mozek je zvyklý na nějaká pravidla a těžce si zvyká na nová. Počítač nemá problém s takovými procesy jako je koordinace, cit pro rytmus apod. Důkladně a přesně zahraje 32 taktů s posunem o jednu dobu (nebo o půl), což je pro živý orchestr velmi náročné až nemožné.
- Poslušnost. Počítač si nikdy nedomyslí nic za nás, je to mechanický stroj, který dělá přesně to, co mu bylo nařízeno. Pokud výsledek neodpovídá předpokladům, problém je vždy na straně programátora a jeho kódu. Počítač nebude vkládat význam do žádného procesu, který provádí.

Existují prakticky dva přístupy, jak přijít na nové věci. Tím prvním je mít od začátku představu o výsledku a hledat systém, jakým lze tohoto výsledku dosáhnout, přizpůsobit mu okolnosti. Druhým přístupem pak je nastudovat dostupné dispozice a okolnosti a následně vyzkoušet, jakého výsledku s jejich pomocí lze dosáhnout. Podle mě většina inovativních objevů vzniká právě tou druhou variantou. Právě proto směřuji svůj nástroj k tomu, aby se stal spíše platformou pro objevy a experimenty, než nástrojem pro realizaci konkrétního zadání. [6]

Gurevich ve své práci považuje otázku kolaborace v hudbě za naprosto fundamentální. Odkazuje na časy před zavedením současných médií a vysílání a před samotným vývojem nahrávací techniky. Tehdy nebyla možnost poslechnout si hudbu jinak, než si ji zahrát nebo si poslechnout živou hudbu dalších lidí. Dnes se jak poslech, tak i tvorba hudby stala převážně individuální disciplínou. Online technologie s sebou nesou spoustu výhod, ovšem je důležité pamatovat a klást důraz na kolaborativní a sociální význam hudby. Tedy nejen na poslechovou, ale i na její tvůrčí funkci [7]

Na trhu elektronických hudebních nástrojů nyní převládají takové, které hudebníkům i nehudebníkům připadají typické. Pro člověka, začínajícího se zvukem, jsou záležitosti jako oscilátory, filtry a složité modulátory minimálně na první pohled těžko pochopitelnou záhadou. Začátečníkům se nevěnuje tolik pozornosti, kolik by bylo zapotřebí. Čas, který začátečník musí strávit nastudováním toho, jak fungují různé ovladače a nespočet neznámých parametrů, je často tak enormní, že ho to může odradit nebo se tím začne nudit a ztratí zájem. Zároveň v současné době můžeme využít faktu, že platformy typu Youtube, Tumbler a další podobné mají na své uživatele jistý kreativní vliv a z pasivních pozorovatelů je mění

na uživatele aktivně generující nějaký obsah. Podporují zájem o tvorbu, který - včetně zájmu o tvorbu zvukovou - rapidně vzrostl. A tato skupina nezkušených uživatelů je pravděpodobně při správně postavené reklamní strategii hlavní cílovou skupinou všech online nástrojů pro tvorbu, obzvlášť kolaborativních.

Nejvýznamnějším aspektem ve vzdělání je podpora, lidský faktor a možnost okamžitě dostávat odpovědi na neustále se rodící dotazy. Nejjednodušším způsobem, jakým toho lze docílit, je vytvořit prostředí, které bude zajímavé jak pro zkušené uživatele, tak i pro začátečníky. Další intuitivní podporu lidem, kteří se prozatím ztrácejí v nespočtu zvukových parametrů, poskytuje adaptabilita uživatelského prostředí a reprezentace ovládacích prvků pomocí interaktivní a abstraktní vizualizace klasických potenciometrů a posuvných faderů pomocí počítačové grafiky. I když je hudba sama o sobě prostředkem komunikace, viz. kapitola o sémantice, nejefektivnějším způsobem, jak něco vysvětlit, vyjádřit své pocity nebo se na něčem domluvit, je stále slovní popis - chat. [8]

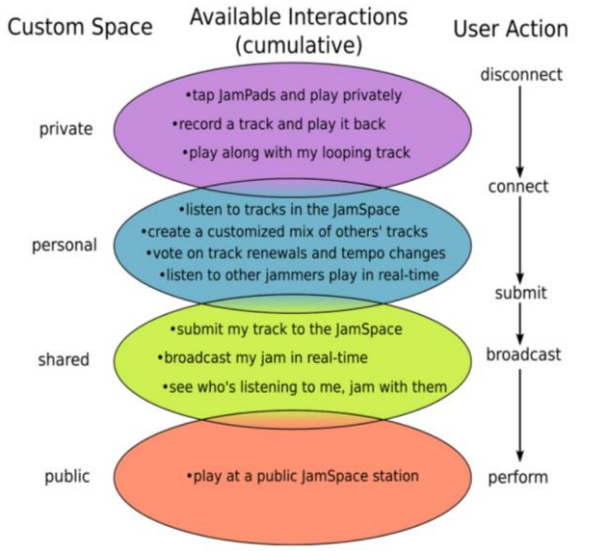
Z průzkumu trhu rovněž vyplývá, že existuje spousta aplikací, orientovaných přímo na hudební produkci. Naopak málo je těch, které jsou věnované zvukovým experimentům. [9] Rozdíl mezi tvořením hudby a zvukovou tvorbou je signifikantní a uživatelské prostředí tomu musí být přizpůsobeno. Tyto dvě kategorie můžeme slovně popsat a odlišit od sebe jako přehrávání zvuků a hraní se zvukem. Můj projekt spadá do té druhé kategorie. Uvedu několik faktorů, které by měly být zohledněné při návrhu uživatelského prostředí pro zvukovou tvorbu:

- vyloučení nebo minimalizace tradičních hudebních prvků, tedy vypuštění notace a vyloženě technických pojmů (noty, stupnice, oscilátory, filtry a tak dále)
- použití metaforické grafiky, znázorňující známé objekty z reálného života, fyzické jevy nebo naopak abstraktní figury a procesy.
- způsoby interakce by měly být zjednodušené a neměly by vyžadovat použití složitých kontrolérů
- dostupnost: možnost provozu na různých platformách (webová implementace je v tomto případě ideální)

Cílem návrháře uživatelského prostředí je dosáhnout správné rovnováhy mezi expresivitou a komplexností nástroje. Dalším prvkem, který může být protikladem

komplexnosti (nebo spíš počtu parametrů - prvků ovládání), je volný prostor pro inovaci a adaptaci ovládacího rozhraní. Především proto, že k novým objevům v oblasti spektra často dochází ne díky tomu, na jaký nástroj hrajeme, ale díky tomu, jak na něj hrajeme, jak jej ovládáme. Škála variability ovládání softwarových nástrojů je rozhodně menší než u reálných nástrojů, ovšem exponenciálně roste se zvýšením počtu ovládacích prvků. Tento jev je nejnázornější u modulárních systémů, kde máme možnost měnit propojení jednotlivých bloků. Kromě prostoru pro inovaci, sloužícího k udržení pozornosti nejen teprve začínajícího publika, ale i zkušenějších zájemců, nesmíme zapomenout na to, abychom ponechali prostor pro sebezdokonalování a zlepšování se v ovládání daného nástroje. Nesmíme také zapomenout na to, že vizuální složka nese v ovládacím rozhraní vždy více informace oproti složce zvukové, navržený interface by tedy neměl svojí grafikou uživatele moc upoutávat, aby ho nevyrušil ze soustředění na aurální vjemovou složku. [10]

Ve své práci JamSpace Gurevich zabíhá v psychologickém rozboru uživatelského prostředí a jeho příjemnosti opravdu do hloubky a definuje několik rovin osobního prostoru - rozděluje ho na 4 části (viz Obr.1): privátní, personální, sdílenou a veřejnou. Odděluje je podle toho, jaké informace (dostupné interakce a činy uživatelů) se s ostatními sdílí a jaké ne. [9]



Obr. 1: Schéma uživatelem konfigurovatelného rozhraní [9]

4. PRŮZKUM EXISTUJÍCÍCH PROJEKTŮ

Téma online kolaborace, online tvorby a online vzdělání je v posledních letech více než populární a jejich rozvoj nezastavitelně stoupá. Vyvíjí se v mnoha směrech, vznikají nové nástroje a prostředí pro vývoj těchto nástrojů. Když mluvíme o hudební tvorbě, tak prvním softwarovým nástrojem, na který si každý pomyslí, bude samozřejmě DAW (Digital Audio Workstation). Výrobců těchto programů je celá řada a dlouhá léta si mezi sebou konkurují. Je logické, že ve chvíli, kdy technologie a technické parametry přenosu a zpracování zvuku dosáhly určité úrovně, všichni se jali vyvíjet nástroje pro umožnění studiového nahrávání na dálku. Výsledkem toho jsou kvalitní produkty téměř od každého z velkých zástupců DAW. Dále uvádím příklady některých z nich. Podívám se také na nástroje, jejichž primárním účelem není přímo nahrávání hudby, ale které staví na kolaboraci nebo online přístupu.

4.1 Ableton Link

Technologie umožňuje synchronizaci tempa, fáze i start/stop mezi několika aplikacemi na jednom nebo více zařízeních. Jako síťové propojení se používá lokální síť, která spojuje ostatní uživatele. Automaticky se vytvoří session, do které se připojíte přes síť. Při příchodu a odchodu účastníků se session nepřerušuje. Program vytváří společnou časovou osu nebo osu událostí (timeline) a ta se adapuje tak, že každý může hrát sám, zároveň ale zůstává synchronizován s ostatními. Ableton Link podporuje i vzájemné propojení mobilních zařízení. [11]

4.2 Avid Cloud Collaboration for Pro Tools

Technologie umožňuje propojení uživatelů Pro Tools online přes síťové úložiště. Jde o sdílení a možnost současného otevření stejného projektu (session) uloženého na online úložišti (cloudu). Jde o sdílení souborů, jako jsou MIDI stopy, editování, změny v mixáži apod. Program je bohužel omezen jen na tři současně pracující uživatele, ovšem počet lidí, kteří mají k projektu přístup, není omezen, to znamená, že se uživatelé mohou střídat. Sdílí se a ukládá se na úložiště celý projekt, ne jednotlivé soubory, proto je přístup dostupný z jakéhokoliv pracoviště. Neexistuje host ve smyslu lokálního ukládání projektu, existuje pouze majitel úložiště, které sdílí svá data s ostatními. Součástí toho režimu programu je také chat umožňující snadnou a nezatěžující komunikaci mezi účastníky nahrávacího procesu. [12]

4.3 NINJAM from Cockos incorporated (REAPER)

Poslední ze skupiny DAW je produkt od developera programu Reaper s názvem NINJAM - open source software umožňující lidem hrát na reálné hudební nástroje společně přes internet. Jde o streamování audia každému z účastníků dané session, takže se všichni navzájem slyší. Proces mixáže a přístupu k němu je vyřešen tak, že každý může míchat jen svůj vlastní mix, to znamená, že každý uživatel dostává sadu mnoha audio stop - multitracku a míchá si ho na svém uživatelském konci. Na úkor přenosového pásma je tím dosažená záloha dat pro každého po ukončení session k dalšímu případnému zpracování a zároveň se tím sníží zátěž procesoru na straně serveru. Streamuje se komprimované audio přes společný NINJAM server. Pro komprimaci je používán formát OGG Vorbis pro jeho nízký datový tok a přijatelnou kvalitu. Pro 8 zároveň hrajících osob je zapotřebí servru, který je schopen přenášet data s minimální výstupní rychlostí 3 mb/s a vstupní rychlostí 600 kb/s. Je dostupná taky funkce uložení nekomprimovaného signálu pro následné kvalitnější zpracování. Způsob synchronizace tohoto programu

se skrývá v kompenzaci zpoždění. S ohledem na zpoždění, které nasbírá audio po cestě přenosu: hardware (nejméně 5 ms), kodek (nejméně 20 ms), průměrné zpoždění síťového připojení (40 ms) - je prakticky nereálné přenášet tímto způsobem real-time. Synchronizace je vyřešena prodloužením této latence o fixní časový úsek (počet taktů). Ve výsledku každý účastník session hraje s posledním "nahraným" časovým úsekem a zároveň nahrává svůj aktuální (budoucí) úsek. Pro samotné hráče je tento způsob hry neobvyklý, ovšem vývojáři tvrdí, že si na to časem lze zvyknout a že jde stále o nejefektivnější řešení. Vývojáři sami uznávají, že je to "zčásti nástroj, zčásti hračka" [13]

4.4 The Reactable

Dalším nástrojem, který určitě stojí za zmínku, je Reactable. Je to fyzický (hardware) nástroj pro interaktivní víceuživatelské tvoření hudby. Interfacem je stůl, na který se dají umisťovat objekty-krychle, představující jednotlivé moduly (generátory, přehrávací moduly, filtry, zpožďovací linky, sekvencery a efektové procesory). Moduly se vzájemně propojují a fyzická manipulace s nimi – například otočení, přemísťování atd. - mění parametry zvuku. Nástroj je kolaborativní, ale není online, i když existuje i jeho softwarová verze pro tablety. Za jedním stolem mohou pracovat nejvíce čtyři osoby, ovšem existuje možnost propojit více stolů mezi sebou softwarovými simulátory a streamem audia. Reactable se dá vnímat také jako kontroler, kterým bezesporu je. Softwarová část Reactable využívá k realizaci technologie podobné těm, které jsou součástí mého projektu. Implementace Reactablu je rozdělena na 4 části: kontrolní, snímací, zvukovou a vizuální. Kontrolní část spojuje ostatní tři a je napsaná v programovacím jazyce Java. Snímací část sleduje pomocí kamery, umístěné nad stolem, polohu a pozici jednotlivých objektů na stole a pravidelně posílá tyto informace hlavní jednotce pomocí OSC zpráv. Zvuková část je implementovaná pomocí prostředí PureData, kde je každý modul realizován jako abstrakce a může se nezávisle modifikovat. Grafika je pak tvořena samostatným programem v OpenGL. [14]

4.5 Multi Web Audio Sequencer

Nejbližším k mému projektu je Multi Web Audio Sequencer. Nástro je na bázi klient-server aplikace na webu, přístupném více uživatelům najednou. Server je opět realizován pomocí Node.js a komunikace pomocí web socketů. Použití web socketů nám umožňuje nestarat se o pořadí provedení operací, přicházejících od více uživatelů najednou, protože to pořadí vzniká přirozeně a je garantované pořadím příchodu zpráv na server. Nástroj ke hraní je bohužel omezen jen na přehrávání samplů podle jednoduchých triggerů, tzn. je to pouze sekvencer. Nástroj je celkem základní, má navíc prvky ovládání hlasitosti, mutování a sólování, které ovlivňují lokální mix, podobně jako u NINJAM aplikace. Jsou tady zohledněné i způsoby komunikace mezi lidmi a to pomocí chatu a přihlášení pod uživatelským jménem. Výhody a nevýhody přihlášení budou podrobněji probrány v kapitole návrhu mojí aplikace. Důležitým prvkem uživatelského prostředí je rozdělení do místností. Omezuje to zátěž na serveru v první řadě tím, že je omezen počet uživatelů, kterým je aplikace přístupná najednou, a ve druhé řadě tím, že jeden uživatel odesílá změny jenom těm adresátům, kteří s ním sdílejí místnost. Tyto informace se posílají pomocí JSON zapouzdřených zpráv. Mrtvé body nebo zablokování systému ve chvíli, kdy se více uživatelů snaží o změnu stejného parametru, jsou vyřešené "zmrazením" nebo blokací tohoto parametru na straně serveru, takže ho ovládá jenom ten, kdo se ho první ujal. [15]

4.6 Soundworks

Dalším zajímavým projektem je Soundworks, jehož framework umožňuje lidem vytvářet nástroje pro kolaborativní hudební vystoupení, kde se účastníci přemisťují v prostoru a používají svůj smartphone ke generování zvuku a světla dotykem a pohybem. Proces používání programu je rozdělen na dvě fáze: nastavení a předvádění. Během nastavení probíhá přihlášení uživatele, synchronizace, umístění v prostoru a kalibrace mobilního zařízení. Druhá část je dána samotnou

akcí a je na programátorovi/na umělci, jak bude probíhat. Tento framework používá také klient-server architekturu a realizován je pomocí Node.js a Socket.io, se kterými se seznámíme v dalších kapitolách podrobněji, protože jsou také součástí mého programu. [8] Drobným specifikem tohoto frameworku je, že si uživatel může vybrat jednu z více rolí, "soloist" - účastník se sólovou partií v hudebním díle, "controller" - účastník, kterému jsou přístupné prvky ovládání globálních parametrů, "shared-env" - role odpovídající za grafickou reprezentaci aplikace a audio rendering. [16]

5. MOTIVACE A NÁVRH

Ačkoli je oblast online kolaborativních hudebních nástrojů docela nová, v současné době už existuje dost nástrojů, na které mohu odkázat. V předchozí kapitole ovšem byly uvedené jenom některé, vybrané dle toho, že používají podobné technologie nebo mají nějaký společný rys, který je součástí nebo i dokonce základem mého projektu. Žádný z nich ale nesplňuje všechna kritéria a mnou vytyčený hlavní účel. Kdybych rekapituloval předchozí tři kapitoly, tak v návrhu své aplikace chci zohlednit následující body:

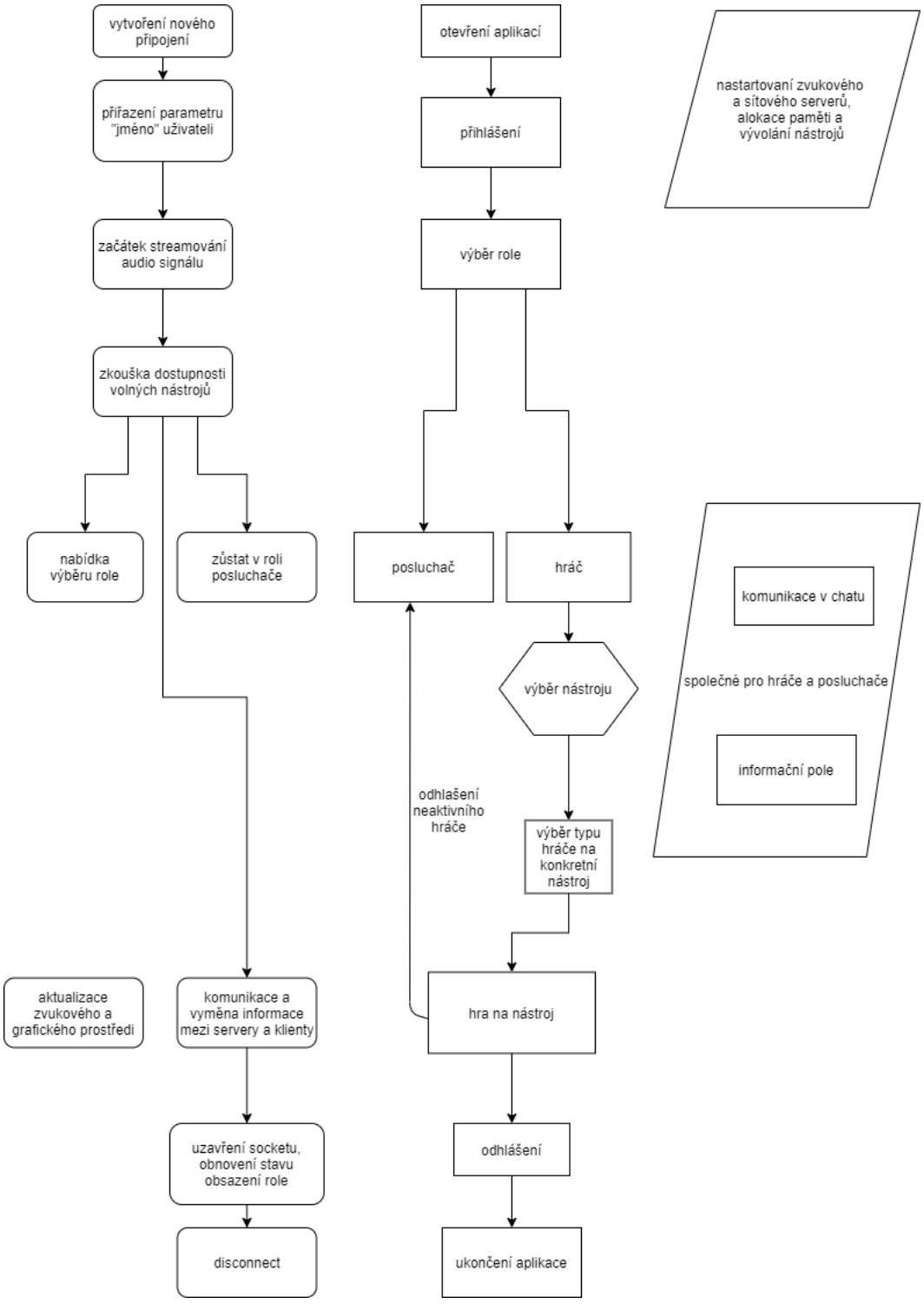
- online přístup formou webové aplikace
- zpracování v reálném čase
- možnost komunikace prostřednictvím chatu
- víceuživatelský přístup (možná dokonce možnost společně ovládat jeden nástroj)
- unikátní grafické řešení a jeho návaznost na zvuk
- inspirativnost jednotlivých nástrojů
- sekundární edukativní charakter
- orientace na barvu a texturu zvuku

Hudba vždy byla obdařena privilegiem vyjadřovat emoce, ale zároveň je prostředkem k mezilidské komunikaci. Mým cílem je vytvořit takové "zvukové hřiště" pro lidi, které by zpříjemňovalo prostředí pro konverzaci. Mělo by tvořit zvukové pozadí pro učení a hledání nových nápadů. Při zamyšlení nad možným cílovým publikem, využívajícím moji aplikaci, jsem si nepoložil otázku, jak oslovit co nejvíc lidí, ale zaměřil jsem se na určitou cílovou skupinu. Jde mi o to, aby lidem byl poskytnut takový nástroj, který potřebují, který hledají a který jim bude sloužit pro objevování nových inspiračních zvuků. Přinejmenším by jim měl otevřit nový způsob, jak přijitna nové zvuky.

V rámci konceptu mého projektu je kontroverzní a sporný prvek přihlášení. Podle Jona Gallowaye [38] proces registrace na začátku může u uživatele vyvolat nepříjemné emoce. Stává se to z důvodu nastavení bezpečnostních údajů na různých

stránkách a také kvůli potřebě věnovat čas registraci na něčem, co v tu chvíli ještě není tímto uživatelem vyzkoušeno. Je to ale nezbytné pro ušetření času, přihlášení usnadní orientaci v místnostech a umožní lepší přehled v chatu. (specifický nickname je vždy názornější než User#111). Zajímavou problematikou vztaženou k přihlášení je otázka identity oproti anonymitě. Je možné vyhovět oběma například tím, že přihlášení zjednodušíme jen na jméno nebo nickname bez hesla - protože nemá cenu vytvářet plnohodnotný účet a uchovávat nějaká data, bez mailu. Nechceme přece uživatele zavazovat spamováním a newslettery, ale chceme jej přilákat zážitkem, za kterým se bude vracet. Ovládání takového nástroje musí být maximálně nenáročné.

Koncept a vzhled aplikace se podobá pokerovým hracím simulátorům. Uživatel se po otevření a přihlášení do aplikace připojí do jedné z místností, kde bude vyzván k výběru experimentálního hudebního nástroje. Dále pak bude zkoumat možnosti tohoto nástroje individuálně i možnosti hraní v kolektivu v rámci místnosti. Podrobný model z pohledu uživatele v kombinaci s paralelní větví, zobrazující procesy probíhající v programu na pozadí, je znázorněn na Obr.2. Sestavení tohoto globálního systémového modelu a schématu komunikace je na Obr.7, kde byla použita služba pro tvorbu uml diagramů draw.io.



Obr. 2: Wireframe aplikace

6. ZAMÝŠLENÉ ZVUKOVÉ MODULY

Představte si krabičku, ve které volně létají částice nebo hmyz a narážejí do sebe i do stěn krabičky. Při nárazech se mění směr jejich letu. Každý náraz generuje krátký zvuk. Bude možné měnit parametry, kterými mám na mysli počet hmyzu, jeho typ a velikost, materiál krabičky nebo její tloušťku. Tyto parametry budou samozřejmě ovlivňovat zvuk intuitivním způsobem: hustotou narážení, místem narážení, dynamikou nárazů, filtrací zvuku.

Jedná se o kombinaci granulární a subtraktivní syntézy.

Možné zlepšení: přidat do krabice předmět, od kterého by se hmyz držel stranou, tím vnést další ovládací prvek.

Dalsim modulem je krabička, do které naléváme tekutiny. Každá z těchto tekutin představuje zvláštní texturu zvuku. Zvuk je plynulý, modulovaný dlouhými komplexními obálkami. Do krabice je možno tekutinu jak přidávat, tak z ní i odebírat. Pokud přidáváme víc tekutiny, než je objem krabičky, zvyšujeme tlak - což také ovlivní zvuk ve formě zkreslení. Krabice může prasknout a vybuchnout, pak začínáme znovu s novou, prázdnou.

Jedná se o kombinaci additivní syntézy a morphingu.

Možné zlepšení: použít k ovládání interní gyroskop mobilního zařízení.

Krabice s nestandardním dozvukem. Simulace těžko dosažitelných až neexistujících podmínek v reálném světě a modelace chování zvuku v těchto podmínkách. Zdroje zvuku jsou klasické oscilátory kvůli připoutání uživatele k vlivu prostředí. Navíc, zkušenější a náročnější uživatel ví, jak by měl znít sinový nebo pilový průběh a dokáže si na základě reálného vjemu a očekávání udělat srovnání.

Parametry k ovládání: vlhkost, výběr různých druhů plynů, tlak a další.

Jedná se o kombinace efektu dozvuku a fyzické modelovací syntézy.

Možné vylepšení: nahrávání svých vlastních zvuků místo klasických oscilátorů.

Krabička, rozdělená na několik oblastí, kdy každá oblast znázorňuje svůj vlastní filtr. Přejížděním myší přes tyto oblasti se bude měnit zvuk. Vizuálně budou vypadat jednotlivé oblasti jako skla (optické filtry) a pohybovat se bude objektem za nimi. Ze stejných důvodů jako v předchozím případě jsou zdrojem zvuku jednoduché generátory.

Jedná se o kombinaci filtrace zvuků a wavetable syntézy.

Možné vylepšení: nahrávání svých vlastních zvuků po přidání více vrstev skla (filtrů do série).

Krabice, ve které jeden uživatel pohybuje předmětem a druhý světlem. Vlastnosti zvuku závisí na světle a míře osvícení. K výběru budou nabízené různé typy světla, jeho intenzita, teplota (saturace), druh atd. Ze které strany přichází světlo poznáme panorámováním.

Jedná se o kombinaci více druhů syntézy a zpracování zvuku.

7. POUŽITÉ TECHNOLOGIE

V této kapitole budou uvedeny jednotlivé technologie, které jsem použil v rámci realizace svého programu. Některé z nich již byly částečně uvedené v předchozí kapitole a to i s příkladem nebo důvodem užití v rámci jednotlivých projektů. Zde každou technologii popíšu podrobněji, uvedu některá historická fakta, která ovlivnila jejich podstatu při návrhu, případně funkcionalitu nebo charakteristiky, kvůli kterým byly použité.

7.1 Web stránky a aplikace

Obecně jsou to stránky, jejichž obsah je odeslán serverem ve formě HTML klientovi jako odpověď na jeho žádost (request). Tento obsah konvertovaný a přečtený prohlížečem, který vytváří DOM (Document Object Model) - sadu pravidel nezávislých na platformě nebo jazyku API, která popisují přístup a zacházení s obsahem strukturovaným do XML a HTML souborů. [17] Důležitou vlastností webové aplikace je možnost sdílení dat, ať už mezi klientem a serverem (nebo s jinými klienty přes server) nebo s dalšími aplikacemi. Další výhoda webové aplikace oproti obyčejné standalone aplikaci je nezávislost na operačním systému. Možnost sdílení dat i nezávislost na operačním systému jsou pro můj projekt klíčové vlastnosti. Původní webové stránky byly statické, ale s rozvojem technologií se teď jejich obsah může dynamicky měnit pomocí JavaScript kódu.

7.2 Javascript

Jedním z programovacích jazyků, který je v oblasti webu bezkonkurenční, je JavaScript. JavaScript nebyl tak dlouho ve stádiu vývoje, aby bylo možno vychytat všechny problémy a drobnosti. Byl vydán v celkem nekompletním, syrovém stavu, přesto si hned získal popularitu, globálně byl přijat a stal se jedním ze základních webových jazyků. Poukazuji na fakt, že jeho popularita nebyla způsobena jeho kvalitou jako programovacího jazyka, ale podmínkami a okolnostmi, za kterých byl vydán. Je vlastně jediným podobným webovým jazykem a dodnes je první volbou pro většinu začínajících webových programátorů. Javascript je samozřejmě stále vyvíjen, mezi jeho aktuální silné stránky patří např.:

- Funkcionální programování
- Loose typing - dynamické typování (ve fázi kompilace není tak přísný k chybám a varováním), což nám dává určitou svobodu, ale zároveň přenáší odpovědnost na fázi testů.
- Object literal notation (způsob vytvoření objektu sepsáním jeho parametrů v poli ve tvaru páru [klic:hodnota] oddělených čárkami). Z tohoto formátu se dále rozvíjel formát JSON.
- Prototypová dědičnost (předávání parametrů objektů od jednoho k druhému)

Problémem JavaScriptu je propojování jeho struktury globálními proměnnými, které potom patří do globálního objektu. [18] Původně byl JavaScript jenom jazykem klientské části webu, ale v roce 2008 přišel Chrome s novou implementací JavaScriptu s názvem V8 a tím připomněl, že se tento programovací jazyk dá použít i na straně serverové. V roce 2009 se objevil projekt Node.js [19]

7.3 Node.js

Platforma na programování webových aplikací a jakýchkoli dalších projektů založených na modelu klient-server. Rozvinutá na principu škálovatelnosti síťových aplikací, asynchronního vstupu/výstupu a událostní architektuře. Událostní architektura představuje princip, kdy program reaguje na konkrétní událost a musí ji nějak obstarat. To ovšem znamená, že ve chvíli, kdy se nic neděje, nemá programátor takovou kontrolu v porovnání s jinými typy programování. Může to vypadat například tak, že uživatel klikne na něco a programátor musí rozhodnout a napsat, co se v této chvíli stane. Dokud ale uživatel nereaguje, tak se nic nezmění

Node není možné označit ani za kontejnerový ani za samostatný programovací jazyk. Je to spíš samostatný virtuální stroj pro programování v JavaScriptu. Hlavním důvodem použití Node je pro nás umožnění běhu javascriptu nezávisle na prohlížeči. Sám o sobě je Node docela základní a omezený software. To co z něj dělá zajímavější a univerzálnější nástroj, jsou moduly - základní bloky, za jejichž pomoci je tvořena Node aplikace. Vlastně každý JavaScript soubor je v podstatě modul. Na začátku kódu je každý modul nahraný nebo vyvolaný v hlavičce pomocí funkcí require, poté jsou jeho funkce přístupné. Správcem těchto modulů je takzvaný NPM – Node Package Manager. Skrz něj jednotlivé moduly snadno vyhledáváme, importujeme a instalujeme. Jsou to jinými slovy knihovny. Ve své práci použiju například: express, supercollider.js nebo p5.js. Seznamy použitých modulů a jejich verze jsou uložené ve zvláštním souboru package.json. Ten také obsahuje další informace o projektu – metadata, a formou tagů nám sděluje například název, cestu, autora atd. Node moduly a npm packages jsou dvě odlišné věci, i když mohou být tím stejným. NPM package je totiž hotový balík, který je připravený k distribuci pomocí npm, zatímco node modul může být pouze část našeho programu uložená zvlášť mimo projekt, na kterou se můžeme odkazovat. Jedná se o mechanismus enkapsulace, zapouzdření. NPM packages jsou uložené ve složce node_modules, uživatelské moduly ale najdeme ve složce samotné aplikace a

při volání pomocí funkce require se na ně odkazujeme odlišně. [20] Jedním z dostupných a při navržení serveru velmi populárních frameworků je Express. [39]

7.4 Express

Framework Express byl použit kvůli jeho minimalističnosti — pomáhá nám ušetřit čas a nemusíme rozepisovat javascript kód, přitom to není na úkor funkcionality, protože jeho další vlastností je flexibilita a možnost doplnit klasický framework dalšími rozšířeními. Zároveň nás zbavuje nutnosti procházet si a učit se kompletně všemu znovu od začátku. Postup je spíš ten, že dohledáváme, co zrovna potřebujeme. Další výhodou Expressu je jeho výpočetní rychlost, která je významná spíš pro velké web aplikace, ale s ohledem na strukturu mého programu může být výhodná i pro mne. Koncept, na kterém je založen Express, se nazývá scaffolding. Při programování každý nový projekt potřebuje ze začátku určité množství kódů, které bývá často podobné napříč projekty - "podle šablony". Express právě tento kód generuje automaticky. Má samozřejmě i své slabé stránky: nemusí vždy podporovat jazyk nebo prostředí, ve kterém programujete. Také vytváří jména proměnných, na která nejste zvyklí, což vede k přepisování kódu nebo k nutnosti zvyknout si na nová pojmenování - obojí zabírá čas, který by naopak měl být ušetřen. U web aplikací můžeme mluvit o dvou typech: client-side aplikace a server-side aplikace. Rozlišují se podle toho, na které straně se renderuje obsah webové stránky, jestli na straně klienta nebo na serveru. V prvním případě se klientovi posílá jenom základní struktura stránky a potom se pomocí JavaScript kódu modifikuje její obsah. V druhém případě se celá stránka vygeneruje na serveru a pošle se klientovi. Často není možné přesně určit, je-li aplikace čistě client-side nebo server-side a tyto hranice jsou dnes víceméně rozmazané a prolnuté, a to i v případě mé aplikace. Express skvěle zvládá oba tyto případy, proto je mou volbou v předloženém programu. [21]

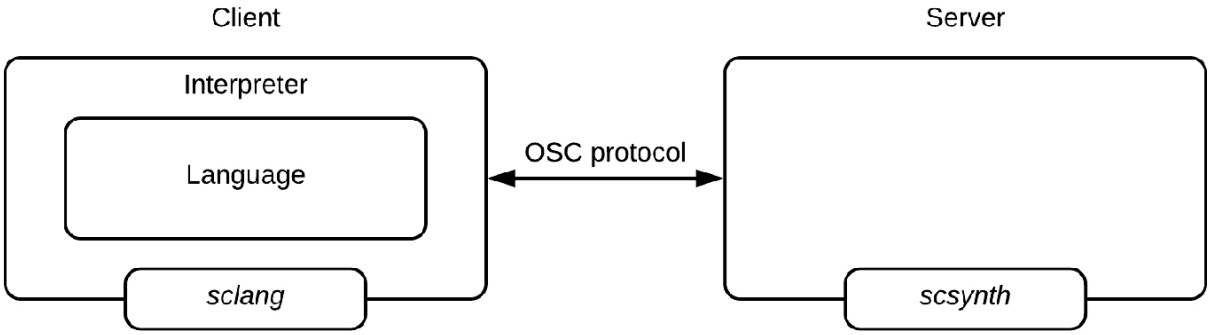
7.5 SuperCollider

SC3 je textový editor, moderní textový programovací jazyk, kompilátor a digitální syntezátor. SuperCollider prošel dlouhým vývojem, počínaje první verzí od Jamese McCartneyho v roce 1996. Pak se třikrát kompletně přepisoval. V poslední, aktuální třetí verzi, se rozdělil na dvě části: the language (sclang nebo klient) a zvukový renderovací systém (scsynth nebo server). Tyto dvě vrstvy jsou propojené protokolem OSC (OpenSoundControl), o kterém jsem se již zmiňoval v předchozích kapitolách. Architektura klientovy části je modelovaná systému Smalltalk, jeho syntax vypadá podobně jako C++. Dokáže přesně vyjadřovat čas a buduje algoritmické bloky pro hudební i jiné časově založené kompozice. Serverová strana byla vyvíjena pro vysoce efektivní zvukový render v reálném čase. Zvukové procesy jsou tvořené grafy syntézy postavené z bloku UGen (Unit Generátor). Signály mohou být propojené přes audio sběrnici nebo přes sběrnici kontrolní. Zvukové soubory a jiná data se ukládají do bufferu. Tato klient-serverová implementace s sebou nese několik výhod. K první patří to, že jiné aplikace mohou použít server k renderování zvuku. Za druhé server může běžet na jiném počítači nebo dokonce na více počítačích zároveň. Třetí výhodou je to, že oddělení klienta a serveru je dělá stabilnějšími a spolehlivějšími. Na druhé straně jsou samozřejmě i nevýhody. Vždycky bude přítomné zpoždění, způsobené síťovou architekturou (je to nepříjemné v případě SuperCollideru, který je cílený a zaměřený na real time zpracování zvuku). Síťové prostředí programu nás nutí k určitému způsobu přenosu informace, který nám přidělává práci při zpracování audio streamu po vzorcích. A poslední nepříjemností je, že nejsme schopni se přímo odkazovat na serverovou paměť ze strany klienta. Programovací jazyk SuperCollider může být taky snadno rozšířen o další knihovny nebo jednotlivé bloky. [22] [6] [23]

SCsynth je serverová část aplikace SuperCollider. Stará se o audio syntézu a zpracovávání. Neumí objektově orientované programování ani SCcode.

Language je jazyk programování SC, který je určen pro uživatele. SClang je pak tvořen dvěma částmi: interpreter a klient. Interpreter zjednodušuje použití the language pro uživatele tím, že ho přeloží na OSC zprávy (viz Obr.3). Tato struktura

dovoluje vykonat části kódu zvlášť. Klient posílá na server sc language kód přeložený interpreterem na OSC zprávy. [24]



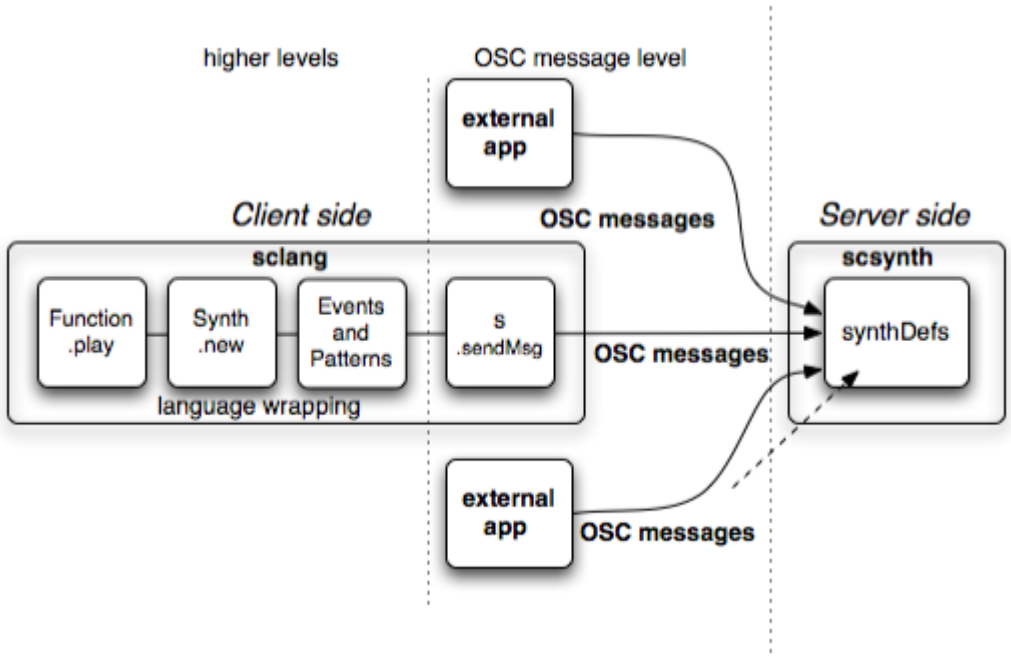
Obr. 3: Struktura aplikace SuperCollider [25]

První, co musíme vždy udělat, pokud plánujeme použít SuperCollider pro práci se zvukem, je nastartovat server. Stačí to udělat jednou a pak můžeme spouštět jednotlivé části kódu a funkce. Zvlášť, lokálně a rychle.

SuperCollider není moc vstřícný ve smyslu debuggingu, často přímo neukazuje, co máte v kódu za problém a jakého typu je chyba, proto se jednoduché chyby jako ztracená závorka, záměna velkého s malým písmenem, čárka na špatném místě nebo její nepřítomnost mohou hledat déle, než je to obvyklé.

Nehledě na to SuperCollider poskytuje řadu nástrojů ke stálému sledování probíhajících procesů. Například zobrazení jádra nodu nebo indikátory vstupu/výstupu. Node je superclass nad synthem a grupou. Node je jednotka na serveru, instance objektu. Jedna instance je synth, několik instancí je možné spojit do grupy a tím usnadnit ovládání několika takových jednotek. Do grupy mohou patřit taky jiné grupy. Právě tento systém je strukturou nodu a můžeme ho zobrazit pomocí funkce *s.plotTree()*. Písmono "s" s tečkou na začátku označuje, že zobrazujeme strukturu serveru uloženého pod proměnnou s. Je to lokální přednastavený server. Synth vytváří instanci na základě SynthDef (viz Obr.4), proto jako první argument má název SynthDef, po kterém následuje pole argumentů. Dva poslední argumenty nám říkají, jak zařadit synth, jestli do grupy nebo zvlášť . Také

sděluje, kam ho zařadit do struktury nodu (na začátek, na konec, po/před nějakým jiným nodem nebo jestli má jiný node nahradit). SynthDef je nejdůležitější a nejsložitější třídou v SuperCollideru - je šablonou pro vytvoření synthu (nodů). Je zapouzdřením pro reprezentaci, uložení a vkládání na server, mimo jiné poskytuje metody pro vytvoření svých vlastních uživatelských argumentů typu SynthDef. SynthDef je konstrukce, která znázorňuje jeden nástroj. Je tvořen UnitGenerátory neboli moduly: generátory, upravovače obálky, filtry, efekty atd.



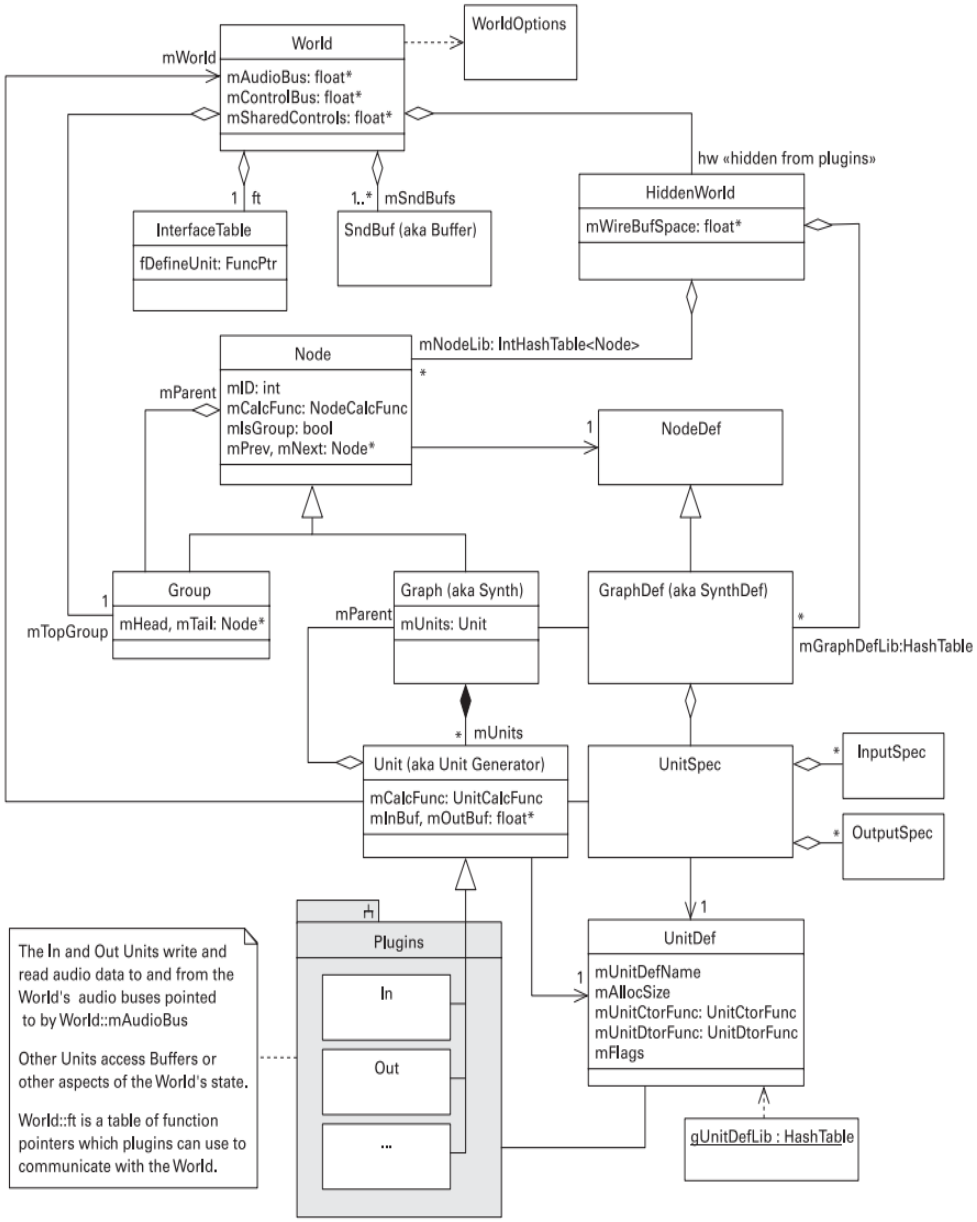
Obr. 4: SClang jako high-level klient [24]

V kódu jsou to funkce, jejich argumenty jsou parametry modulů. Způsob, jakým jsou jednotlivé moduly propojené, popisuje instance funkce UGen Graph Function, která je jádrem SynthDef. Tato funkce také obstarává obsluhu vstupů a výstupů včetně správy parametrů určených k vnějšímu ovládání. K obsluze vnějšího ovládání používáme UGen s názvem Control a funkci set. Většinou se ale o tento UGen nemusíme vůbec starat, vytváří se sám ve chvíli, kdy přepisujeme jiným UGenum metodu *.kr* (control rate) nebo *.ar* (audio rate). Výstup Ugenu v SuperCollideru je vždy počítán po celých blocích vzorků. Audio rate má standardní

hodnotu 44100 vzorků za sekundu nebo 64 vzorků v bloku SC, control rate vychází na jeden vzorek na blok. V případech, kdy na vstupu některých UGenu je vyžadován signál obsahující control rate, audio rate signál prochází také, ale za relevantní so považuje pouze první vzorek z balíku. V opačném případě se chybějící vzorky dointerpolují (doplní se střední hodnoty, aby výsledný signál působil více "spojitě"). Takovéto převody je možné realizovat na uživatelsky potřebných místech pomocí funkcí *A2K()* a *K2A()*. Skoro u každého UGen členu narazíte na argumenty mul a add, násobení a sčítání. Tyto argumenty jsou definované za účelem implementace násobení a sčítání různých signálů, které se mohou na dané vstupy dostat. V audiu se často setkáváme s různými rozsahy hodnot například 0 až 1, -1 až 1, mínus nekonečno až nekonečno, mínus nekonečno až 0, 0 až nekonečno atd. Pomocí argumentů mul a add dokážeme rozdíné rozsahy kompenzovat. Dalším takovým rozdílem může být reprezentace signálu v logaritmickém a lineárním měřítku. SC řeší i mapování těchto rozsahů, slouží k tomu funkce *linexp()* a *explin()*. Pro adaptaci různých rozsahů jednoho měřítka bez použití násobení a sčítání použijeme linlin. Argumenty pro tyto funkce jsou totožné: vstupní signál, dolní a horní hranice původního a a dolní a horní hranice výstupního signálu. [23] Audio a Control busses jsou sběrnice, které nejsou nic jiného než krátkodobá paměť, slouží k propojování signálů, stejně tak jako sběrnice nebo kanály na mixážním pultu. Jejich počet je stanoven při spuštění serveru, standardní hodnota je 128 pro audio sběrnici a 4096 pro sběrnici kontrolní. Z těchto 128 prvních sběrnic je 8 rezervovaných pro hardware a kanály od 8 do 15 jsou určené pro čtení vstupu ze systému. Zbylé jsou volné pro uživatelské použití. Nevylučuje se ani použití prvních 16, jenom se to obecně nedoporučuje pro zachování běžné architektury programu a udržení spokojenosti programátora. Toto výchozí vymezení prvních 16 sběrnic platí jak na vstupech, tak na výstupech, zobrazení se vyvolá metodou *s.meter*. Počet sběrnic je samozřejmě možné kdykoli zjistit a změnit jej pomocí metody *.numInputBusChannels* pro vstupní sběrnice (a obdobných metod pro výstupní sběrnice, audio i kontrolní) v rámci ServerOptions, která je vytvořena pro každý server zvlášť a ukládá a spravuje jeho parametry.

Hierarchie a propojení výše zmíňených základních bloků je znázorněna na Obr.5.

Obdobným způsobem je možné stanovit i maximální počty SynthDef bloků a Nodů, což nám pomůže omezit přístup na server a zabránit jeho přetížení. Podrobněji se o tomto zmiňuji v dalších kapitolách o riziku a testování programu.



Obr. 5: Vrstevní diagram významných částí struktury SuperCollideru [23]

7.6 Processing a p5.js

Processing je textový programovací jazyk a prostředí (PDE - The Processing Development Enviroment) pro generování a zpracovávání obrazu. Úspěšně balancuje mezi jednoduchým, přehledným softwarem pro začínající umělce a efektivním nástrojem pro programátory a vizuální designéry. Pracuje jak s vektorovou grafikou, tak i s 3D modely, s různými druhy integrovaných rozhraní (ovládání myší a klávesnicí), se síťovou komunikací a objektově-orientovaným programováním. Díky své struktuře je vhodný i pro účely výuky, jak pro výuku počítačových věd, tak i pro výuku práce s obrazem. Vizuální zobrazení pomáhá umělcům rychleji pochopit základní principy programování a umožňuje jim tedy realizovat jejich nápady s použitím jiného způsobu myšlení. Ti, kteří jsou zběhlí v kódu a logice počítačových procesů, mohou rozvíjet své estetické představy pomocí pro ně známé syntaxe. Syntaxe je inspirována Javou a některými C metodami. Processing je open source program, což v tomto případě primárně znamená stálý a nezastavitelný rozvoj a neustálé rozšiřování dostupných knihoven a funkcí. Processing byl už od začátku vyvíjen jako open source, takže vznikl velmi rychle a bez velkých finančních výdajů, proto je nyní distribuován zdarma. PDE (prostředí processingu) je celkem jednoduché a obsahuje: textový editor, konzolu a tlačítka se základními funkcemi. Každý spuštěný program se zobrazuje vždy v nově otevřeném okně. Programům se říká Sketch a mimo hlavního souboru mohou obsahovat i různá mediální data, použité knihovny apod. Exportovaný program pro otevření mimo PDE se zabalí a překonvertuje do java kódu a kompiluje se v Java aplikaci. U tohoto procesu nepůjdeme moc do hloubky, protože to není nutné - my použijeme trochu jinou formu tohoto programu, konkrétně p5.js. P5.js je knihovna Processingu pro programovací jazyk JavaScript, jedná se tedy o jednoduchou a vhodnou možnost, jak dostat moje Sketche a grafické návrhy na web.

Každý kód v Processingu je rozdělen na dvě funkce, *setup()* a *draw()*. Toto rozdělení umožňuje programu běžet kontinuálně a zároveň vytvářet animace a realizovat interaktivní programy. Na začátku programu je jednou spuštěna funkce *setup()* a potom se stále dokola ve smyčce opakuje volání funkce *draw()*, které

vykreslí vždy jeden frame (snímek). Samotné vykreslení proběhne vždy, když program dorazí na poslední řádek. Rozsáhlejší programy obsahují samozřejmě i dodatečné uživatelské funkce, třídy i objekty. [27] [26]

7.7 Komunikace

Abych zvolil vhodnou technologii pro realizaci svého serveru, zvukového a grafického rozhraní, musel jsem učinit další krok - nalézt vhodný způsob komunikace. Grafika se generuje přímo na klientově straně a zvuk zase na serveru, ovšem na jiném, než na kterém běží samotné jádro aplikace - je to tak nejvýhodnější pro realizaci real-time aplikace. K propojení paralelně běžících částí aplikace potřebujeme komunikační nástroje, které budou jak kompatibilní s vybranými prostředími, tak budou schopné fungovat v rámci webového rozhraní aplikace. Klasické možné způsoby komunikace přes web jsou:

- Request/Response - nejzákladnější způsob pocházející z principu webu jako takového. Klient pošle žádost na server a dostane odpověď, na základě které znovu vygeneruje stránku, případně její část (pomocí AJAX)
- Polling - pužíváme jej ve chvíli, kdy potřebujeme obnovit obsah stránky bez klientovy interakce, například vypsat výsledky fotbalového zápasu. V té chvíli posíláme požadavek na překreslení jednou za určenou dobu a to nezávisle na tom, zda proběhla nějaká změna nebo ne.
- Long polling - jednou za čas se otevírá kanál komunikace na delší dobu. Je to ve své podstatě real time komunikace, ale je nutné znát dopředu přesný čas, kdy chceme komunikaci navázat.

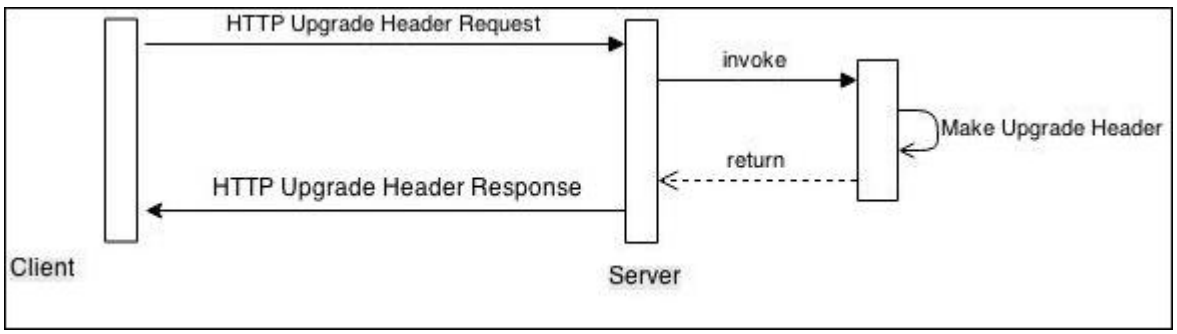
7.7.1 Web sockets

Real-time komunikace bylo vždy těžké implementovat, programátoři používali Flash pro předávání dat, ovšem to bylo neefektivní a výpočetně příliš náročné řešení, které se prakticky přestalo používat ve chvíli, kdy HTML přišel s obnovenou verzí 5, jejíž součástí bylo i Websocket API. Toto API nám dává možnost přestat používat externí pluginy a zároveň mění i samotný princip komunikace. Namísto konstantního bombardování serverů informacemi o provedených změnách otevíráme přímý komunikační kanál a přímo posíláme data ze strany klienta i serveru. Navíc víme, kdy která zpráva dorazí.

Vlastnosti websocket komunikace:

- oboustranná komunikace
- bezpečný přenos
- rychlá odezva (50-150 ms)
- malá spotřeba přenosového pásma
- umí komunikovat přes TCP/IP
- podpora téměř všech existujících prohlížečů

Spojení se navazuje pomocí HTTP mechanismu handshake (viz Obr.6), ovšem samotná komunikace probíhá ve formě malých packetů ve vrstvě TCP protokolu a není proto zobrazována v rozhraní prohlížeče pro developery.[28]



Obr. 6: HTTP handshake [28]

7.7.2 Socket.io

Socket.io je jedna z javascriptových open-source knihoven běžících na Node.js. Realizuje oboustrannou real-time komunikaci na bázi websocketů. Tato knihovna pro nás vytváří spolehlivější API než při použití čistě websocketů a navíc se stará i o ty případy, kdy ztratíme websocketové spojení. V ten okamžik ho plynule nahradí technologií long polling, o které jsem se zde již zmiňoval. Navíc samostatně testuje kompatibilitu programu s asi 25 webovými prohlížeči, což nám dokáže ušetřit čas do budoucna.

7.7.3 OSC protokol

Podle Matthew Wrighta je OSC protokol ‘a protocol for communication among computers, sound synthesizers, and other multimedia devices that is optimised for modern networking technology’ [29]. Tento protokol byl vytvořen za dvou důvodů. Jednak proto, aby hudba nezůstávala uzavřená na jednom místě a bylo možné v tomto rychlém světě informačních technologií navazovat hudební kolaborace na dálku - např. aby se počítače mohly propojit s hudebními nástroji. Dále proto, že již existující MIDI protokol nebyl dostatečně pochopitelný a otevřený. OSC proto do názvu zprávy nedává jako midi čísla, ale slovní neboli znakový popis. Technicky tak sice obětujeme nějaké bajty navíc, ale vzhledem k rychlostem, jakými je nyní možné komunikovat, se jedná o zanedbatelné množství. Technické limity tohoto protokolu si popíšeme později.

OSC protokol blíže nespecifikuje, jaká technologie bude použita pro přenos, informace jsou přenášeny ve formě balíků nebo zpráv, které obsahují adresu, type tag a argument. Adresa teoreticky může obsahovat jakoukoliv informaci, je ovšem primárně určena ke směřování neboli rozpoznávání zprávy. Zapisuje se fromou adresy přes lomítko a vypadá například následovně: ‘nastroj1/zdroj/freq/par’. Type tag nese informace o typu dat obsažených v jednotlivých argumentech. Je podporováno téměř cokoliv od floatu a integeru, přes boolean po string, jako extra

typy dat mohou být například kódy barev, zprávy ve dvojkové soustavě. Argument je pak zpráva jako taková, její samotný obsah. [30]

7.7.4 Napojení zvuku

Na straně klienta zatím máme jenom grafiku a zvukové prostředí supercollider je propojeno se serverem jenom pro komunikaci, zvuk nám totiž stále zůstává na jeho serveru. Z variant, které jsem zkoumal, nejlepšími řešení bylo webRTC a IceCast.

WebRTC je v základu protokol pro komunikaci a přenos informace, včetně video a audio složky, v reálném čase. Je určen primárně pro chaty, messangery, video rozhovory a podobně. Jeho nevýhoda pro nás je, že používá peer-to-peer spojení. My naopak potřebujeme vysílat více lidem z jednoho serveru. Znamená to, že průměrná rychlost, hypoteticky v řádech desítek KB/s, se vynásobí počtem připojených uživatelů. Tím pádem v případě 1000 uživatelů rychlost přenosu a nárok na výstupní datový tok internetu je 40 MB/s, což je nepřijatelné pro budoucí provoz. Existují řešení, rozšiřující tuto technologii o mezičlánek-server, například Janus, který přijímá tok webRTC z našeho serveru a následně ho překóduje a vysílá každému, kdo se k němu připojí.

Právě na podobném principu funguje i IceCast, jenom nepotřebujeme nic navíc mezi tím. IceCast je server pro vysílání multimédia. Když se podíváme na jeho vstupy, tak akceptuje hlavně drivery pod operačním systémem linux jako jsou ALSA (Advanced Linux Sound Architecture) nebo OSS (open sound system). Muj system běží na platformě Windows, tento problem nám vyřeší externí program butt (broadcast using this tool), který podporuje víc operačních systémů a jeho účelem je vysílat audio data z mikrofonu nebo vstupu počítače do IceCastu. Sám o sobě serverem není (tím je ale Icecast), podporuje samozřejmě stejné zvukové formáty jako IceCast, umí nahrávat, při správném nastavení se automaticky připojuje k serveru a v případě výpadku dokáže se napojit znovu sám.

Když se podíváme na možnosti výstupu serveru SuperCollideru, tak uvidíme nástroj pro směrování signálu uvnitř systému na OS Linux s názvem JACKaudio.

Pro OS Windows najdeme taky dlouhý seznam driveru včetně těch nejrozšířenějších: ASIO,WASAPI,MME atd. Potřebujeme teď přesměrovat výstup z SC na Butt. JACKaudio má windows verzi, není bohužel kompatibilní s SC. Obdobným programem na Windows je VB-Cable: Virtual Audio Cables. Umožňuje propojit vstupy a výstupy různých aplikací uvnitř OS.

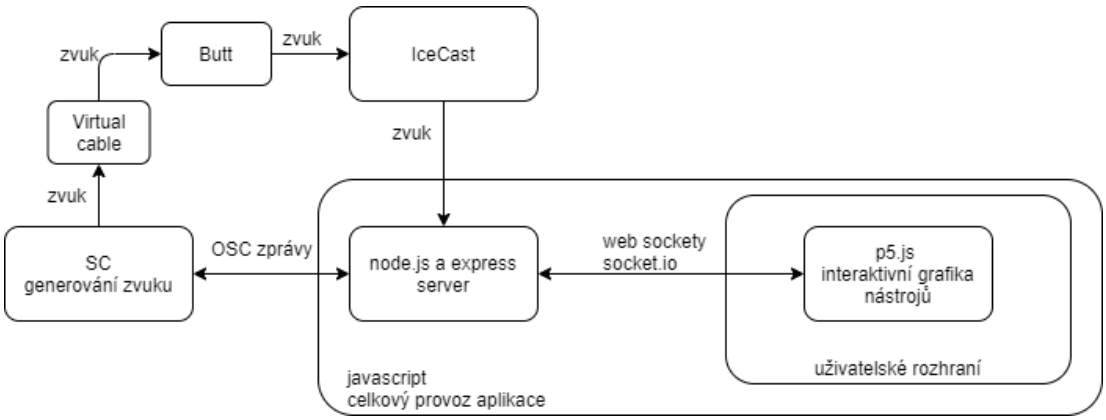
V celém systému je důležité dohlížet na stejnou vzorkovací frekvenci, bitovou hloubku a formát audio signálu.

Vytvořil jsem tak řetězec, který dostane zvuk ze serveru SuperCollideru na webové stránky k uživateli.

Vícekanálový přenos nebo přenos více stop každému uživateli pro individuální ovládání jejich hlasitosti v této konfiguraci není možný.

Zmiňoval jsem roli mixéra, která v průběhu vývoje aplikace se jevila jako zbytečná a dokonce nevhodná, protože je nelogické, aby jeden uživatel ovládal hlasitost - primární parameter ostatních a byl například schopný vypnout někoho úplně. Příliš by to ostatním kazilo zážitek. Dávat přístup všem je taky nesmyslné, protože by vznikl nepořádek. Proto jsem úrovně všech nástrojů vyrovnal tak, aby s ohledem na charakter zvuků bylo možno slyšet každý nástroj.

Rekapitulací této kapitoly je model uvedený na obr.7, který popisuje celkový systém komunikace hlavních bloků programu.



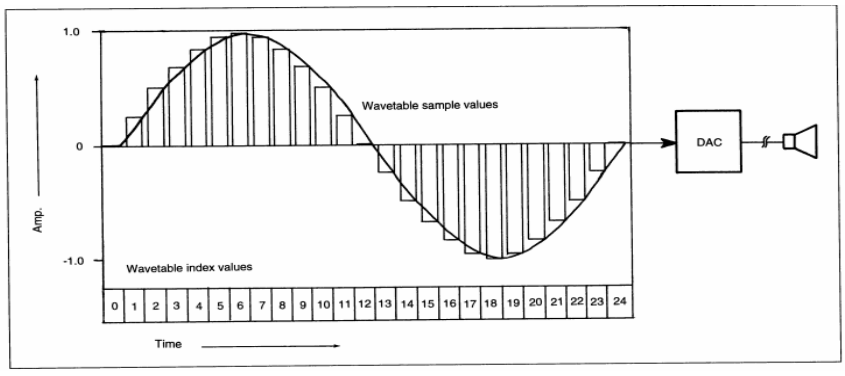
Obr. 7: Komunikační model hlavních bloků aplikace

8. TEORETICKÝ ZÁKLAD

V této kapitole stručně popíšu základy signálového zpracování a různé techniky, které budu používat v rámci svého projektu, ať už přímo nebo jen k jeho popisu.

8.1 Číslicový zvukový signál

Zvuk je v počítači jako jakákoliv jiná informace reprezentován jakýmsi tokem číselných hodnot a pro jejich popis se většinou používá dvojková soustava. Hlavními vlastnostmi číslicového zvukového signálu jsou vzorkovací frekvence, udávající počet vzorků, které máme k dispozici pro popis průběhu jedné sekundy signálu. Dále je to bitová hloubka, udávající přímo počet úrovní, kterých může signál nabývat (rozlišení dynamického rozsahu, s tím je nerozlučně spojena často zmiňovaná úroveň kvantizačního šumu). Pro přehrávání zvuku se tyto vzorky jeden po druhém přečtou a odešlou na výstup číslicově-analogového převodníku (DAC - digital analog converter) viz. Obr.8, který je řízen vzorkovacím časovačem a přeměňuje tok čísel na úrovně napětí. Ve finále je signál v analogové podobě rekonstruován pomocí filtru typu dolní propusti (filtrace vysokofrekvenčního šumu a vzorkovací frekvence). [31]



Obr. 8: Přehrávání číslicového audio signálu [31]

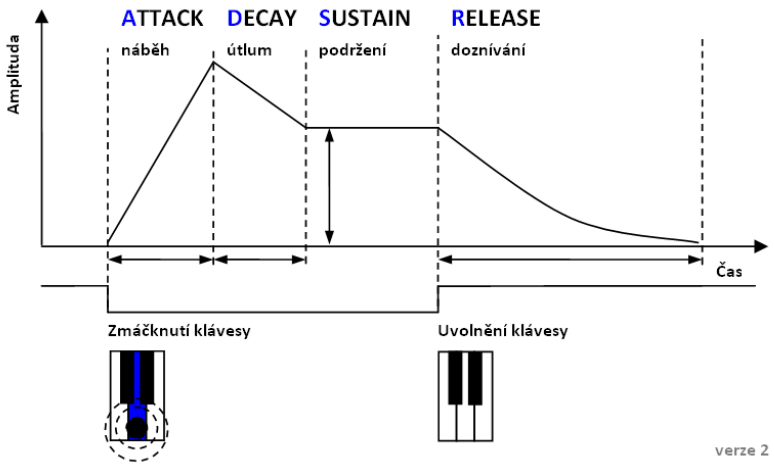
Při číslicové zvukové syntéze je tento tok čísel generován pomocí matematických algoritmů a následně je na základě různých estetických kritérií a požadavků dále transformován a zpracováván.

V rámci syntézy nebývá technicky efektivní generovat primární signál po jednom vzorku. Výhodnější bývá využít periodicity většiny hudebních signálů a ukládat do tabulek nebo zásobníků jednu periodu signálu nebo jeho krátké úryvky a následně je skládat, přehrávat a manipulovat s nimi. Tomuto procesu se věnuje tzv. table-lookup syntéza.

8.2 ADSR

Při jednoduchém generování zvukového signálu v počítači zůstává jeho amplituda konstantní, zvuky jsou ale zajímavější, pokud mají amplitudu v čase proměnnou - toho lze docílit modulací výstupu generátoru. Nejčastěji používaným modulátorem pro amplitudu je ADSR obálka. Standardně je rozdělena na 4 fáze (viz. Obr.9):

- Attack (náběh, nástup): udává, za jakou dobu se signál po stisknutí klávesy dostane na své maximum. Okamžitý náběh je typický pro perkuse. Smyčce mají naopak většinou náběh pozvolný.
- Decay (útlum): doba, za kterou signál poklesne ze svého prvotního maxima na ustálenou hodnotu.
- Sustain (zadržení): udává, jak dlouho signál zní při stálém držení klávesy.
- Release (uvolnění, doznívání): doba do úplného doznění tónu po uvolnění klávesy [32]



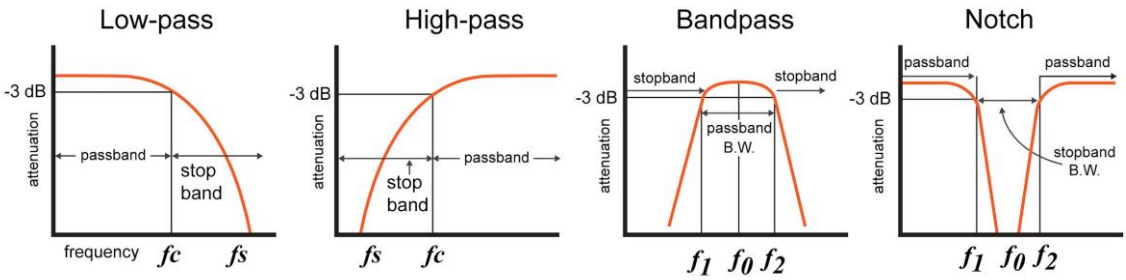
Obr. 9: Průběh ADSR obálky [32]

Existují i složitější nebo naopak jednodušší tvary obálek s jiným počtem fází a lineárními i nelineárními průběhy.

8.3 Filtry

Součástí skoro každého zvukového systému jsou filtry. Liší se podle typu (viz. Obr.10), základními jsou:

- dolní propust
- horní propust
- pásmová propust
- pásmová zádrž

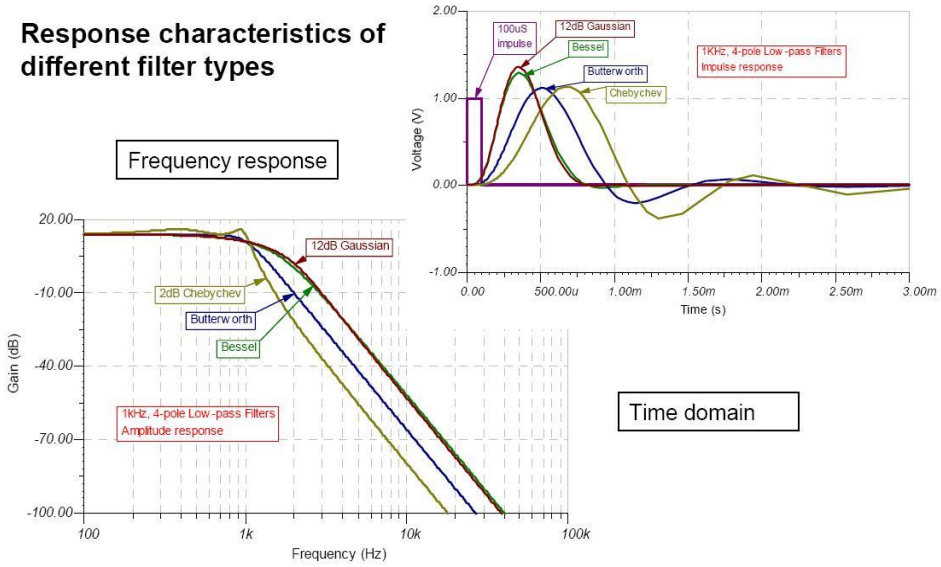


Obr. 10: Základní typy filtrů [33]

Tyto filtry lze tvořit na základě různých aproximačních matematických modelů dle požadavků na jejich kmitočtové charakteristiky (viz Obr.11), např.:

- dle Chebysheva
- dle Bessela
- eliptická aproximace
- dle Gausse
- dle Butterwortha

Liší se zvlněním, sklonem nebo zesílením v propustném a nepropustném pásmu.



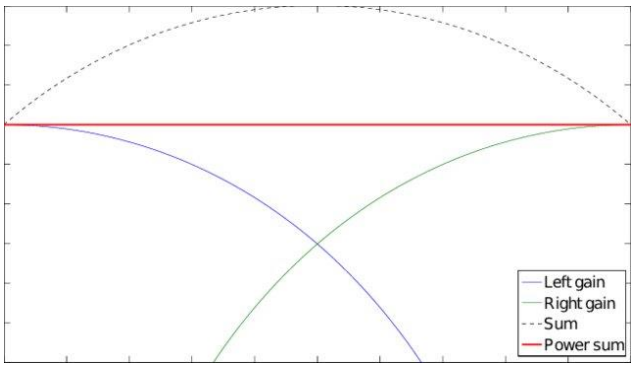
Obr. 11: Kmitočtové charakteristiky a impulsní odezvy filtrů [34]

Pro popis daných filtrů používáme nejčastěji následující parametry:

- Zisk v propustném pásmu
- Útlum v nepropustném pásmu
- Mezní/střední kmitočet
- Činitel jakosti
- Zisk na mezním/středním kmitočtu
- Fázový posun na mezním/středním kmitočtu
- Sklon filtru

8.4 Panoráma, stereobáze

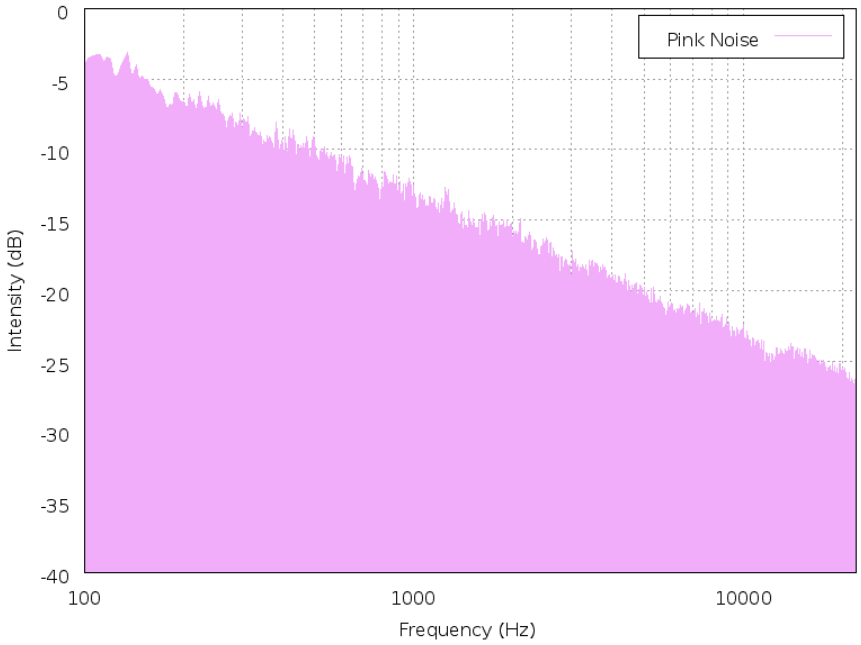
Jelikož pracujeme primárně s dvoukanálovým zvukovým materiálem, je nutné zvolit způsob, jakým budeme přistupovat k panorámování - umístění zvukového signálu ve stereobázi. Rozhodl jsem se pro metodu tzv. *equal power panning*. Signály levé a pravé strany se při panorámování touto metodou neprolínají ve stejném poměru, tzn. nepočítá se čistá suma L+R (na obrázku zobrazeno jako "Sum"), ale počítá se s odmocninou z lineárního rozložení uprostřed stereobáze, jelikož se jedná o stereo - dva kanály, jde o odmocninu z jedné poloviny (~=0.707). Tento výpočet zajistí to, že ať už umístíme pomocí panorámy zdroj kamkoli do stereobáze, budeme tento signál vnímat podobně nahlas. [35] Přechod centrální polohou je tak jemnější a víc odpovídá reálnému očekávání (viz Obr.12).



Obr. 12: Porovnání lineárního a equal power panorámování [31]

8.5 Sonifikace tření v prostředí

Pro sonifikaci tření v prostředí jsem použil metodu postupného přidávání šumu. Zvolil jsem růžový šum, jehož specifikem je rozložení energie ve spektru s exponenciálním poklesem směrem k vyšším kmitočtům (viz Obr.13), tj. konstantní pokles energie na oktávu - takový pokles je podobný tomu, jakým způsobem funguje lidský sluch, proto vnímáme růžový šum jako "víc hudebně vyrovnaný", plný a přirozený.



Obr. 13: Spektrum růžového šumu - logaritmická frekvenční osa [36]

9. REALIZACE

Základ programu je tvořen trojím samostatným prostředím. SuperColliderem, generujícím zvuk, p5.js, knihovnou pro Javascript tvořící grafiku a Node.js, programem pro vytvoření a správu serveru. Tyto jednotlivé bloky spolu komunikují pomocí OSC protokolu při komunikaci SC-Node a pomocí WebSocketu při komunikaci Node-p5. Tuto konfiguraci vidíme na schématu na Obr.5. Podíváme se podrobněji na každou z těchto částí na příkladu prvního nástroje.

Grafické prostředí je rozděleno na dvě části, plátno neboli krabička s pohybujícími se objekty a panel obsahující klasické prvky pro ovládání jako jsou slidery, tlačítka, rozbalovací seznamy a zaškrtávací políčka. Plátno s objekty plní nejen funkci zobrazení, ale zároveň hraje i roli ovladače. Pomocí klasických sliderů a tlačítek ovlivňujeme fyzické parametry virtuálního prostředí a objektů, například: koeficient tření, počet objektů, sílu gravitace, druh materiálu atd. V oblasti plátna pomocí myši přidáváme a odebíráme jednotlivé objekty.

Grafická realizace sliderů a tlačítek již je přístupná v p5.js, ale není zatím vyvinuta na dostatečnou úroveň, proto bylo potřeba doplňovat k jednotlivým ovládacím prvkům popisky. Pomocí funkce createElement() v jazyku p5.js můžeme integrovat tyto popisky přímo ve sketchovém souboru za použití syntaxe HTML, tagy tvořící tento jazyk jsou argumenty zmíněné výše funkce p5.js. K popisu nemohu využít interní funkce text(), protože by její výsledek měl být v prostoru canvasu (plátno s objekty) a slidery jsou mimo tuto oblast. HTML popisky ale na rozdíl od textu v p5.js nemohou jednoduše poskytovat dynamickou informaci o aktuální hodnotě parametru, který ovládáme, proto jsem zvolil kompromis a umístil tyto informace na okraj canvasu.

Pozadí je tvořeno gradientem, kterého jsem dosáhl postupným vykreslováním horizontálních čar, jejichž barva je dána interpolaci dvou barev mapovaných na rozsah plátna. Alpha kanál této barvy je zároveň ovlivněn parametrem friction. Vizuálně je tak tvořen efekt “mlhy”, “závoje” a jednotlivé objekty pak působí jako

náhodně se vyskytující. Pomocí parametru friction je tak znázorněno nastavené tření, které je zvukově navázáno na parametr úrovně přidaného šumu.

Druhým parametrem prostředí je gravitace. Při vzrůstající hodnotě gravitace se na pozadí objevuje oblouk znázorňující planetu. Tento parametr neovlivňuje zvuk přímo, ale upravuje pohyb objektů tím, že začínají být přitahovány k dolní části krabičky.

Zvuk je tvořen primárně nárazem objektů na sebe, s tím souvisí, že další zvukové parametry jsou vázané na fyzické vlastnosti objektu: hlasitost nárazu je přímo úměrná rozměru objektu, parametr hollow (objem duté části uvnitř objektu) přímo ovlivňuje rezonanční schopnosti daného objektu a také se dutý objekt stává prázdnějším, lehčím. Už ze slovního popisu této fyzické vlastnosti je možné odhadnout, že se jedná o filtraci. Použil jsem klasickou horní propust se sklonem 12 db/okt. Graficky je tento parametr znázorněný změnou hodnoty alpha kanálu barvy objektu, neboli průhlednosti daného objektu.

Dalším parametrem společným pro všechny objekty je pružnost. Vizuálně pružnost ovlivní to, že kuličky se neodrazí hned, ale nejprve se trochu stlačí (prolnou se) a až potom se vydají opět na cestu v opačném směru. Zvukově se měkčí náraz projeví prodloužením nástupné a sestupné hrany amplitudové obálky (ADSR).

Další blok na ovládacím panelu je věnován přidávání a ubírání většího počtu objektů najednou. Obsahuje jeden slider, který udává počet objektů, které by měly být přidány nebo odebrány, dvě tlačítka pak spouští celý proces přidávání a odebírání.

Poslední funkcí je zapnutí a vypnutí režimu collapse. Tento režim způsobí, že objekty, které do sebe narazí, zmizí. Řídí se zaškrtávacím políčkem a jednou proměnnou typu boolean, která odpovídá stavu collapse. Pravdivost této proměnné se ověřuje podmínkou ve funkci nárazu collide a je-li collapse true, funkce zruší do sebe narážející objekty.

Zmáčknutí levého tlačítka myši mimo objekty v oblasti zobrazovacího panelu způsobí vznik nového objektu. Způsobí to poloha myši a aktuálně nastavené parametry na ovládacím panelu. V případě, že uživatel klikne na nějaký již stávající objekt, zmáčknutí myši způsobí jeho zaniknutí.

Jádrem programu je pole balls objektu ball. Funkce *setup()*, která je součástí každého souboru p5.js a je volána jednou při spuštění programu, obsahuje inicializaci prvků GUI se žádanou polohou, stylem, výchozími hodnotami a rozsahy. Dále mimo samotného generování objektů vykresluje zmíněné popisky některých prvků, vytváří pracovní plochu, určuje barvu pozadí a posílá prvotní pár bloků komunikace, které budou popsané dále. Počáteční hodnoty jsou definované ještě před funkcí setup. Vytvoření počátečního pole objektů je realizováno cyklem for a přidáváním dalšího prvku pole v každé iteraci. Nové objekty se v programu tvoří dvěma způsoby, náhodně (při spuštění programu a při přidávání více objektů najednou) nebo na základě parametrů zadaných na ovládacím panelu (při kliku na zobrazovacím panelu). Náhodné generování parametrů probíhá variacemi příkazu *random()* s různými argumenty jako rozsah, maximální hodnota, pole možných parametrů (materiál) a výška nebo délka zobrazovacího panelu. Při účelném vytvoření objektu se každý parametr buď načítá z aktuální hodnoty odpovídajícího grafického prvku pomocí metody *.value()* nebo se vypočítá z aktuální polohy myši pomocí funkcí mouseX a mouseY (horizontální a vertikální souřadnice). Po přidání nebo odebrání každého objektu se obnoví počet objektů a aktualizují se indexy stále aktivních objektů (parametr objektu ball). Podle indexu objektu odebíráme z pole konkrétní objekt nebo ho identifikujeme při výpočtech nárazů a pohybu. K odebrání objektu jsem použil metodu *splice()*, protože nevytváří nové pole a neobsahuje právě smazaný prvek. Odebírá totiž prvky přímo ze vstupního pole, které se v tu chvíli stává zároveň i polem výstupním. Při přidávání a odebírání více objektů a obnovení indexace ošetřujeme maximální počet objektů, aby nedošlo k pokusu obnovit parametr již neexistujícího prvku pole.

Aby nedocházelo k přidávání objektů při zmáčknutí myši během ovládání parametrů mimo zobrazovací panel, jsou tyto případy ošetřené porovnáním aktuální polohy myši a mezních hodnot plátna. Rozhodnutí o přítomnosti objektů v oblasti kliku se provádí funkcí porovnávající aktuální polohu myši s aktuální polohou každého objektu se započtáním jeho rozměrů. Barva objektu se přiřadí ve funkci *colorAssign()* podle materiálu, jedná se o jednoduchý switch operátor.

Druhou základní funkcí p5.js sketche je *draw()*, která se na rozdíl od *setup()* opakuje stále dokola. V mém případě obsahuje některé zobrazovací metody a

obsluhuje několik komunikačních cest. Nejdůležitějším blokem je ovšem spuštění funkcí collide, move a display pro každou instanci třídy ball.

Třída ball obsahuje konstruktér s výše zmíněnými parametry, ke kterým se přidává parametr zrychlení, který používáme pro úpravu pohybu.

Funkce collide v cyklu zjišťuje, jestli se některé z objektů prolínají. Děje se to porovnáním jejich vzdálenosti a minimální vzdálenosti, dopočtené ze souřadnic a rozměru objektů. Následně se stanoví nový směr pohybu pomocí trigonometrických funkcí a také určí zrychlení objektů na základě nastavené hodnoty pružnosti.

Započtení vlivu gravitace a odrazu na pohyb se provádí ve funkci *move()* tím, že se upraví rychlost a směr pohybu přičtením nebo násobením daných koeficientů.

Komunikace pomocí web socketů a OSC protokolu probíhá v několika fázích. Její osnova je v souboru index, který po načtení potřebných knihoven vytvoří server. Dále se pro komunikace serveru s klientem založí nové spojení a v rámci něj se vždy pro jeden kanál otevírá nový socket. Zároveň se přidělí nové udp porty pro příjem a odesílání OSC zpráv. Rovnou se nastaví odposlech zpráv ze SuperCollideru, jejich výpis s podrobnými informacemi a přeposílání klientovi.

```
udpPort.on("message", function (oscMsg, timeTag, info) {
  console.log("An OSC message just arrived!", oscMsg);
  console.log("Remote info is: ", info);
   OSCfromsc = oscMsg;
   io.sockets.emit('fromsc', oscMsg['args'][0].value);
});
```

Podíváme se na příklad odeslání zprávy o nárazu.

Odeslání zprávy na straně klienta vypadá následovně:

```
function sendCollide(mater,pruhl,rozm) { // Vstupni
argumenty jsou material, pruhlednost a rozmer.
var dataCollide = { // Vytvoreni zabalovaci promenne pro
prenos
```

```
  x: mater,
  y: pruhl,
  z: rozm
 };
 socket.emit('collideMat', dataCollide); // odeslani
zpravy na server po kanalu collideMat ve forme promenne
dataCollide
}
```

Na serverové straně přijímáme zprávu následujícím způsobem:

```
// vytvoříme odposlech pro kanál a určíme funkci, která
nám říká, co musíme provést se zprávou, kterou dostaneme
socket.on('collideMat',
       function(dataCollide) {
// přebalíme zpravu do jineho tvaru, ktera bude čitelná
pro SuperCollider, tj. OSC zpravy a bude obsahovat
adresu a argumenty v parech s jejich datovymi typy.
        {
  var msg = {
    address: "/hello/collide",
    args: [
      {
        type: "s",
        value: dataCollide.x
      },
            {
                 type: "i",
        value: dataCollide.y
            },
            {
```

```
                type: "i",
        value: dataCollide.z
            }
    ]
  };
   console.log("Preposilam COLLIDE do sc", msg.address,
msg.args, "to", udpPort.options.remoteAddress + ":" +
udpPort.options.remotePort); //vypiseme do konzole co a
kam posilame
  udpPort.send(msg); // a posleme to do SC
  };});
```

Některé zprávy od klienta na server musí být přeposlané zpátky ostatním připojeným klientům pro zajištění víceuživatelského přístupu a ovládání. K tomu slouží příkaz:

```
socket.broadcast.emit('sliderSpring', hodnota); //
prikaz preposlani hodnoty slideru
```

Pokud jeden z uživatelů změní polohu slideru, změní se tato poloha všem připojeným uživatelům - samozřejmě musí být tato operace ošetřena i na klientské straně. Princip je stejný jako příjem zprávy na straně serverové. Pro zajištění tohoto spojení se musíme k serveru taky připojit a to pomocí metody connect a uvedením adresy:

```
socket = io.connect('http://localhost:3000');
```

Localhost je adresa počítače, ze kterého se spouští program. Je ekvivalentní IPv4 verzi 127.0.0.1 [21]

V prostředí zvukového serveru SuperCollideru pro příjem OSC zpráv vytvoříme dvě globální proměnné, kde jedna bude přidělovat port podobným způsobem jako na serveru node:

```
~testNetAddr = NetAddr("127.0.0.1", 57121);
```

Druhá vytvoří objekt pro odposlech zpráv a jejich zpracování:

```
(
~listener = {|msg, time, replyAddr, recvPort|
// blok zpracování
};
thisProcess.addOSCRecvFunc(~listener);
)
```

Blok zpracování OSC zpráv třídí vstup podle adresového stromu ve tvaru '/slider/spring' a dynamicky mění parametry běžících Synthů pomocí metody *set()*. Při nárazu (například) posíláme trigger pro obnovení přehrávání a obálky, spolu s vedlejšími parametry jako amplituda, frekvence a rezonance filtrů, attack a release a další.

Zároveň je nutné při změně parametrů zkontrolovat a upravit rozsahy a to pomocí funkce *linlin()* a dalších převodních funkcí dříve popsaných.

Generování zvuku je řešeno přehráváním zvukových nahrávek sonicky odpovídajícím vybraným materiálům.

V prvním kroku pro tyto soubory alokujeme paměť bufferu na serveru s ohledem na vzorkovací frekvenci, délku vybraných zvuků a počet kanálů (mono/stereo/vícekanálové).

V druhém kroku načteme soubory uložené na serveru a zakládáme SynthDef.

Ten má řadu argumentů, které potom zajišťují ovládání jednotlivých instancí této “šablony”, konkrétně:

```
arg out=0, bufnum=0, rate=1, trigger=1, startPos=0,
loop=0, trig=0, freqf=1000, rez=1, amp=1, atk=0.2,
rel=0.5;
```

Jako obálku jsem použil segmentovanou obálku, protože u ní můžeme zvlášť nastavovat amplitudu a trvání každé z fází a také jejich počet.

Pro stereo reprezentaci jsem použil UGen Pan2, pro jemnější a přirozenější nastavení polohy ve stereo bázi jsem zvolil metodu equal power panner.

Samotné přehrávání probíhá V UGenu PlayBuf. Přehráváme od začátku beze smyčky s poměrem 1.

Ke zvukové reprezentaci tření v prostředí jsem použil generátor růžového šumu.

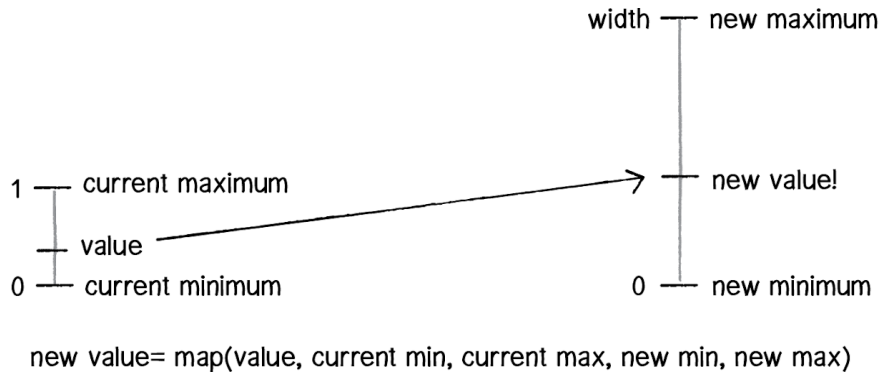
9.1 Water app

Koncepce této aplikace v globálním pojetí je mix různých druhů signálů a zaměření na jejich variabilitu a propracovanost. Aplikace je představena virtuálním akváriem, ve kterém jsou tři tekutiny symbolizující každá jeden zdroj zvuku. Tři uživatelé ovládají každý svoji tekutinu a její parametry, mohou si tak hrát s dílčí částí aplikace, zkoumat charakter vybraného zdroje zvuku a zároveň si ho zařazovat do celku akvária a tvořit tak společnou atmosféru. U aplikace existují stavy, kdy akvárium může být plné. Nehledě na to uživatelé stále mohou vodu přidávat, tím ale zvyšují tlak. Při dosažení maximální hodnoty tlakového indikátoru akvárium praskne a běh aplikace se zastaví. Kromě tlačítka restart se poté veškeré funkce stanou neaktivními až do opětovného zmáčknutí tohoto tlačítka restart.

Pohyb hladiny vody je naprogramován pomocí Perlinového šumu. Tento algoritmus byl pojmenován po Kenu Perlínovi, vynálezci tohoto šumu. Účelem

tohoto šumu bylo vytvoření textur do filmu Tron, za což dostal Perlin ocenění od Filmové Akademie a Akademie věd. Perlin noise může být reprezentován jak v jedné, dvou, tak i ve třech dimenzích. Použil jsem 1D variaci v kontextu jednotnosti 2D grafiky v celé aplikaci. 2D variantu jsem se pokoušel použít pro zobrazení textur jednotlivých tekutin, abych tomu přidal objem prostoru a odlišil druhy tekutin od sebe. Ovšem vypočítávání tohoto algoritmu po jednom pixelu v matici bylo příliš zatěžující na počítač, proto jsem musel přejít na statické textury, které ve tvaru perlinového šumu už nebyly natolik realistické. O tomto podrobněji níže. K rozhodnutí pro výběr tohoto typu šumu mě vedla jeho schopnost napodobovat přírodní textury a to díky náhodnosti, která je do určité míry kontrolovatelná. Toto kontroverzní tvrzení se zakládá na dvou tvrzeních. První je, že použití jednodušší formy perlinového šumu samo o sobě se nechová tak komplexně, proto se většinou sčítá několik jeho instancí s různými frekvencemi a amplitudami. [40] A druhé zní, že opakovatelnost nemusí být zaznamenaná divákem kvůli šumovému chování a transformování. [41]

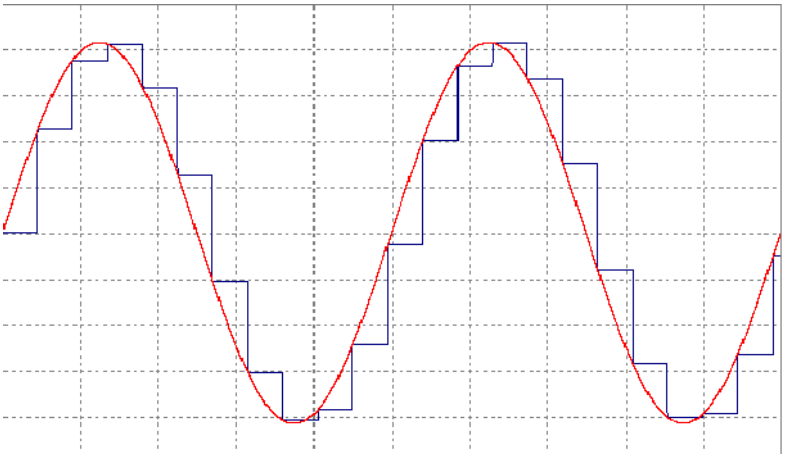
Na výstupu máme šum v rozsahu +-1, který následně mapujeme, podobně jak přicházející OSC zprávy do Supercollideru, v tomto případě na dvě hodnoty, maximální a minimální pro úroveň dané tekutiny.



Obr. 14: Mapování rozsahu [47]

Tyto konstantně generované hodnoty vytváří vlnící se čáru a spolu s čárami kolem okrajů vykreslují zaplněné akvárium. Už zmiňované textury vody byly

vybrány statické, ovšem s ohledem na náhodně se generující povrch jejich vykreslování jsem musel realizovat zobrazením předem nahraných obrázků a to pomocí úzkých čar po 10 pixelech. (viz. Obr.15)



Obr. 15: Vyplnění vodní hladiny po segmentech na principu diference

Proto při extrémních hodnotách parametrů vlnění jsou vidět ostré okraje. Menší hodnoty šířky úseku způsobují velkou výpočtovou zátěž. Aproximace a interpolace těchto úseků by byla možná, ale v rámci algoritmu zbytečně složitá. Jedním z dalších parametrů, které můžeme ovládat, je hustota/průhlednost vody. Ke grafické průhlednosti jsem použil funkci *tint()*, která však je taky dost náročná na výpočet, proto ty výpočty jsou provedené jen jednou při spuštění aplikace. Konkrétně, sada nahraných obrázků textur vody je zobrazena postupně, zpracovaná funkcí *tint()*, načež je uložena. Při změně parametrů jsou vyvolané zpracované obrázky z této sady, proto se průhlednost mění skokově. Při nalévání a vylévání vody kolem kurzoru padají nebo vzlétají kapky uložené v poli droplets a jsou vykreslované a rozpohybované funkcí drawDrop*()*. Tvar kapky je sestaven z jednoduchých linií a matematických funkcí. Barva kapek se přebírá od objektu vody.

Zvuková část pro každou tekutinu je řešena zvlášť, ačkoli všechny tři případy dodržují stejnou logiku. Supercollider má v sobě moduly chaotických generátorů, jejichž použití v oblasti zvuku není obvyklé. Poskytují nám posloupnost pseudo-

náhodných čísel a zvukově se chovají přibližně jako šum. Jsou pseudonáhodné, protože dávají náhodné výsledky v rámci stanovených rozsahů podmínek a jsou určené každý svou konkrétní rovnicí. Tyto posloupnosti jsou velmi citlivé a závislé na změnách počátečních podmínek a hodnot zvolených konstant těchto rovnic.

Těchto chaotických generátorů je v SuperCollideru implementováno přes desítky.

Po poslechovém testu a zapojení Ugenu do zvkového řetezce jsem se omezil na využití Lineárního congruentálního chaotického generátoru jako zdroje signálu.

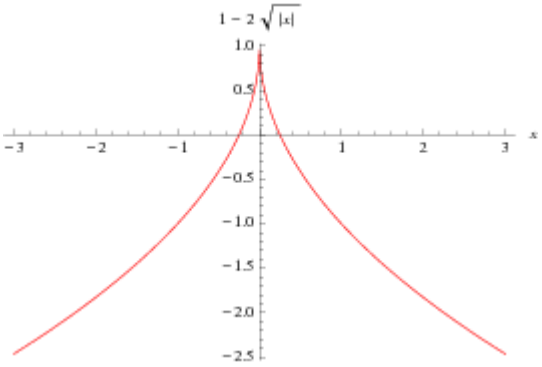
Lineární interpolovaný zvukový generátor založený na diferenciální rovnici:

$$x(n+1) = (a * x(n) + c) \% m$$

a Cuspově mapovanému chaotickému generátoru, který je taky lineárním interpolovaným zvukovým generátorem, avšak založeným na jiné diferenciální rovnici a to:

$$x(n+1) = a - b * \mathrm{sqrt}(|x(n)|)$$

Graficky je znázorněna na Obr.16.



Obr. 16: Charakteristika Cuspového mapování

Tyto dva generátory jsem zvolil proto, že se chovají víc přirozeně, šumově nebo jejich přechody jsou měkčí a plynulejší. Některé generátory jsou těmto podobné, některé znějí velmi digitálně, proto jednoduše nebyly vhodné.

Samy o sobě nakonec zvukové generátory nedávaly uspokojivý výsledek ani jako šum, ovšem dokázal jsem najít intervaly hodnot pro každou proměnnou rovnici

těchto chaotických generátorů tak, aby byly užitečné jako modulační signál. Jako nosný jsem použil sinusový signál a moduloval jsem jeho kmitočet a fázi. Modulace není vždy přímá a jednoduchá, někde je použita v kombinaci s dalšími generátory, například i pro standardní signály (trojúhelníkový průběh nebo pilovitý).

K tomu, abych dostal jemnější a zajímavější detaily, které tyto chaotické generátory produkují, jsem následně použil řetězec z kompanderu a limiteru. Charakter tónu jsem zvýraznil dvoupólovým rezonančním filtrem s nulou v 1 a -1. konstantním zesílením a proměnnou šířkou pásma a frekvencí. Tento filtr byl implementován podle K. Steiglitze. [48]

Součástí řetězce zpracování je taky reverb, u každého SynthDefu je použit jiný a nejzajímavějším je algoritmický "JPverb" reverb, navržený k produkci dlouhého doznívání s efektem choruse. Je inspirován klasickými modely od Lexiconu a Alesisu. Je součástí projektu DEIND. Pro zkreslení signálu při zmíněném zvedání tlaku v akváriu, jsem použil decimátor. Tento modul snižuje bitovou hloubku vstupního signálu a efektivní vzorkovací frekvenci. Nevýhodou je, že zpracovaný signál zní víc digitálně, než klasický distortion, oproti tomu má stabilnější chování a zachovává amplitudu výstupního signálu.

Třetí tekutina se odlišuje od prvních dvou tím, že není tak silně zpracována a představuje spíš tónový generátor několikrát komplexně modulující sám sebe v čase. Tím je zajištěna bohatost spektra a zároveň melodická část.

Poslední parametr, který nebyl popsán, je bloom – tedy vodní květ. Je graficky znázorněn zvýrazněním zelené složky barvy vody a zvukově přidáváním harmonických a vyšších frekvencí. Změna úrovně vody samozřejmě se odráží v hlasitosti jednotlivých zdrojů, ale taky na jejich frekvencích. Každý SynthDef jsem se snažil naprogramovat tak, aby byl dostatečně modulován a nevedl k monotónnosti a statičnosti, která by unavovala posluchače. Ovšem samogenerující systém to není.

Rozbití akvária spustí jednoduché přehrávání předem nahraného samplu.

9.2 Dollar app

Tato aplikace je určena pro dvojici a její princip je podobný aplikacím využívajících sonogram a kreslení na spektru. Pozadí této aplikace tvoří bankovka, na které jeden z uživatelů kreslí vodoznak. Tato kresba ještě sama o sobě zvuk negeneruje, ale představuje data, definující budoucí generování zvuků. Tato data se „přehrávají“ v dalším kroku, kdy druhý uživatel tyto nakreslené tvary vypaluje světlem. Vlastnosti světla a míra osvícení (saturace a svítivost) a taky místo a plocha tohoto osvícení mají přímý vliv na zvuk. Zásadním problémem při realizaci grafiky tohoto nástroje bylo vytvořit několik vrstev, na kterých by se dalo kreslit (vypalování je taky kreslení) a restartovat, obnovovat plochu. P5.js pracuje s canvasem jako hlavní plochou a createGraphics() metodou pro vytvoření dodatečné grafiky. Zvykem je kreslit na hlavním canvase a dodatečnou plochu používat pro zobrazení něčeho jiného. S ohledem na to, že pláten na kreslení jsem měl přes sebe víc, tak ani jedno z nich na hlavním canvasu zůstat nemohlo kvůli obnovení pozadí. To bylo nutné, aby světlo se mohlo pohybovat a přitom zůstat taky “nenakreslené” na pozadí. Process vypalování je realizován tak, že s každým vykreslením procházím matici s NxN prvků (N odpovídá osvícené ploše) kolem polohy kurzoru. Pak dostávám pomocí funkcí get() hodnotu barvy pixelu a porovnávám ji s barvou, kterou používám pro kreslení. Důležité je porovnávat barvy po složkách, modrou s modrou a tak dále, protože proměnná, obsahující pole hodnot barev pro kreslení, se nemůže porovnat s proměnnou pole, které dostaneme z funkcí get(), a to ani po převedení na stejný typ a rozměr polí. Pokud tyto barvy jsou shodné, tak v tomto bodě na hlavní ploše (ze stejného důvodu jako světlo) podsvicují kresbu prvního uživatele pro druhého. Pokud druhý uživatel přitom drží tlačítko myši, tak zároveň na dodatečné grafice vykreslují vypalované body. Jejich barva závisí na saturaci a velikost na intenzitě. Barvy jsou předem definované dvě, červená a žlutá. Pak jsou následně mapované na rozsah slideru saturace. Zvukově tento parametr je realizován rezonančním filtrem *Stresson()*, který přidává šumově-perkusnímu charakteru nástroje určitou tonalitu a další vrstvu. Při vykreslování vypáleného bodu se zpráva posílá. Ty zprávy se neposílají pro každý bod, protože rozlišení obrázku je 800*350

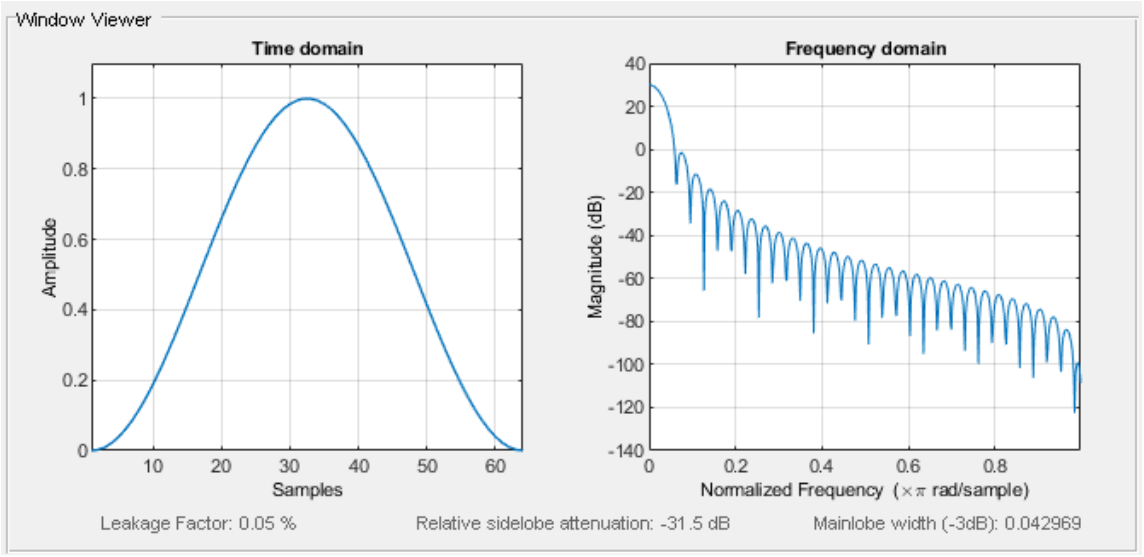
pixelů a pro jednotlivou vrstvu 800*70 pixelů, tj 56000 pixelů a potenciálních grainů, což je příliš moc. Maximální počet grainů v modulu Supercollideru je nastavitelný. Původně a docela rozumně je nastaven na 512. S ohledem na to, že celou bankovku prakticky nikdy nikdo nevymaluje, jsem zkrátil počet odesílaných pixelů na 1/10 horizontálně a ⅕ vertikálně. Ve výsledku na 80*14=1120 pixelů. Pixely však mohou být přemalované, znamená to, že se mohou vypalovat už vypálené body na bankovce. O tom, že uživatel vypálil maximální počet grainů v jedné z oblastí (512) se dozví na serveru Supercollideru. Tuto informaci ověřujeme jednou za vteřinu pomocí objektu *Routine*, který se používá k načasování událostí, v našem případě opakování smyčky. Uvnitř této smyčky je funkce s podmínkou, ověřující nastal-li přesah maximální hodnoty grainů za sekundu. Pak se odešle zpráva klientovi do p5.js, kde se to uživateli zobrazí vykreslováním upozornění na bankovce.

Ještě než upřesním parametry přehrávání, musím konkretizovat, že jádrem zvukové části je modul GrainBuf - granulární syntezátor přehrávající samply z bufferu. Samply jsou předem upraveny na stejnou délku a hlasitost. Je použitá sloka volně dostupného projektu po stopách (stems) z knihovny Cambridge university [42]. Horizontální osa bankovky reprezentuje čas a vertikální je rozdělena na pět dílů, kde každý odpovídá jedné z pěti stop, podobně jako v DAW softwaru. Všechny vlastnosti grainů jsou náhodně rozložené pomocí metody *LFNoise()* a maximálních a minimálních rozsahů stanovených podle maximálních a minimálních hodnot přicházejících z p5.js sketche od klienta. Zkoušel jsem taky přesněji rozložit náhodnost se započtením váhových koeficientů metodou rozdílu bližšího maxima a střední hodnoty. Kvůli různým rozsahům dat tekoucích v toku zpráv od klienta, včetně exponenciálně rozložených a záporných, to vedlo ke zbytečně složitější logice, náročnější na výpočet v počítači. Ty vlastnosti závisí nejvíc na poloze (výška a začátek přehrávání) a pak na saturaci a intenzitě (délka a témbr). Výjimkou je obálka grainů samotných. Ta je stejná a používá se Hannovo okno. Rovnice popisující toto okno je nasledujici:

$$w(n)=0.5(1-\cos(2\pi nN)),\ 0\leq n\leq N,$$

kde délka okna: $L = N + 1$. [49]

Při hodnote L = 64 jeho časové a frekvenční průběhy vypadaji jako na Obr.17.



Obr. 17: Hannovo okno v časové a frekvenční doméně

U této aplikace jsem musel zavést hodně globálních proměnných a to z toho důvodu, že některé výpočtové operace a převody se musely provést v části kódu, která přijímá OSC zprávy a nešlo to tam předat jiným způsobem, navíc tyto hodnoty se měly potom posílat do SynthDefu, proto to bylo celkem zřejmé řešení.

Pro kolaboraci hráčů dvou rolí u této aplikace a konkrétně pro jejich orientaci jsem použil panorámování. Ve chvíli, kdy jeden hráč kreslí, už vypálené granule zní vychýleně ve stereo prostoru v závislosti na tom, kde ten první uživatel kreslí. Kromě toho, že podsvicující může prohlížet a prosvicovat celou bankovku postupně, snadněji a rychleji dostane info o tom, kde najde nakreslené tvary.

9.3 FilterApp

Další nástroj je inspirován syntezátorem na platformě tabletu iOS od firmy Moog s názvem Animoog. [43] Animoog je polyfonický syntezátor s vizuální plochou, kde pohyb po ní umožňuje přecházet od jedné ke druhé zvukové vlně. Stručně je to ve své podstatě wavetable syntéza s originálním zpusobem modulovat signál.

Mou myšlenkou bylo vytvořit Filtertable syntézu. To znamená, že zvukový signál postupně prochází od jednoho filtru k druhému a jejich kombinacím. Použil jsem podobný princip přechodu mezi jednotlivými filtry jako ve zmíněném syntezátoru_Animoog. Hlavní plocha nástroje je rozdělena na několik oblastí, kde každá oblast znázorňuje svůj vlastní filtr. Vizuálně vypadají jednotlivé oblasti jako skla neboli optické filtry. Přejížděním myší přes tyto oblasti se bude měnit zvuk.

Aplikace je určena pro dva uživatele, kde jeden ovládá filtry a druhý pohyb objektu. Objekt se pohybuje mezi majáky, které se dají umisťovat do kteréhokoliv místa v tomto prostoru. Majáky jsou číslované a objekt přechází od jednoho k druhému postupně po trase. Když se dostane k poslednímu, tak přechází na první a takhle ve smyčce dokola. Přechod jsem naprogramoval jak přímočarý, tak i složitější pomocí exponenciálních, kvadratických a dalších matematických průběhů. Lineární přímočarý pohyb mi však připadal nejpřirozenější a pro uživatele logičtější, proto jsem ve výsledku zůstal u něj.

Pro filtry a majáky jsou vytvořeny třídy. Třída *maják()* obsahuje jenom konstruktér se dvěma parametry souřadnic polohy a identifikátorem pořadí a funkcí zobrazení.

Konstruktér třídy filtru se taky neobejde bez souřadnic a identifikátorů, kromě toho ale obsahuje navíc proměnné odchylky, překrývání a táhnutí, kterých se využívá v odpovídajících funkcích. Například při posunu filtru zmáčknutím klávesy se ověřuje, jestli je v oblasti myší vůbec nějaký filtr. Pokud ano, tak proměnné hodnotě táhnutí se přidělí hodnota “pravda”. Aby filtr nepřeskakoval prostředkem do polohy myši, ale pronásledoval ji. Jeho souřadnice se v každém kroku aktualizuji podle souřadnic myši se zachovánou odchylkou od centra.

Pro obě třídy jsou vytvořena pole, kam se ukládají jejich objekty. Těch se potom využívá při pohybu hlavního objektu. Při startu aplikace se vygeneruje vždycky osm filtrů rovnoměrně poskládaných vedle sebe. Jsou čtyři druhy filtrů, které se vizuálně liší barvami. Vygenerované při startu jsou vždy i čtyři majáky s náhodnou polohou. Pohybující se objekt doletí k nultému majáku a pokračuje tak, že u každého majáku vypočítá cestu do dalšího. Hlavní rychlost pohybu se dá měnit. Ovšem čas, za který urazí cestu od jednoho bodu do dalšího, je nezávislý na vzdálenosti. Proto čím dále je cíl, tím rychlejší je přechod. Tak uživatel může

korigovat rychlost pohybu po přímce například počtem majáků rozestavěných na jednotlivých úsecích cesty. Kromě pohybu po trase od jednoho majáku k dalšímu je implementován taky pohyb kolem bodu pohybu. Kmitající pohyb ve dvou osách při velkých rychlostech může vypadat jako rotační. Tato rychlost vychýlení se dá taky ovládat.

Při vytvoření příliš velkého počtu majáků můžeme vždy očistit jejich pole tlačítkem restart, které zanechá jenom jediný bod a od něj můžeme budovat novou cestu. Tlačítko restart postaví do původního stavu i filtry.

Při tahu filtrů jejich průhlednost se mění a tak poznáme, který zrovna v tuto chvíli přenášíme. Pokud je jich pod kurzorem více, tak potáhnou všechny.

Funkce flyOver() každý snímek kontroluje v cyklu podle identifikátoru nad kterými filtry se objekt pohybuje, přičemž sestavuje pole aktuálních fultrů. Toto pole se potom posílá přes server do Supercollideru. Pokud je filtrů nad objektem víc než jeden, jsou potom zapojeny sériově.

Zdrojem zvuku jsou ostatní aplikace a klidně jich může být víc najednou. Funguje to jako externí efekt pro tyto aplikace.

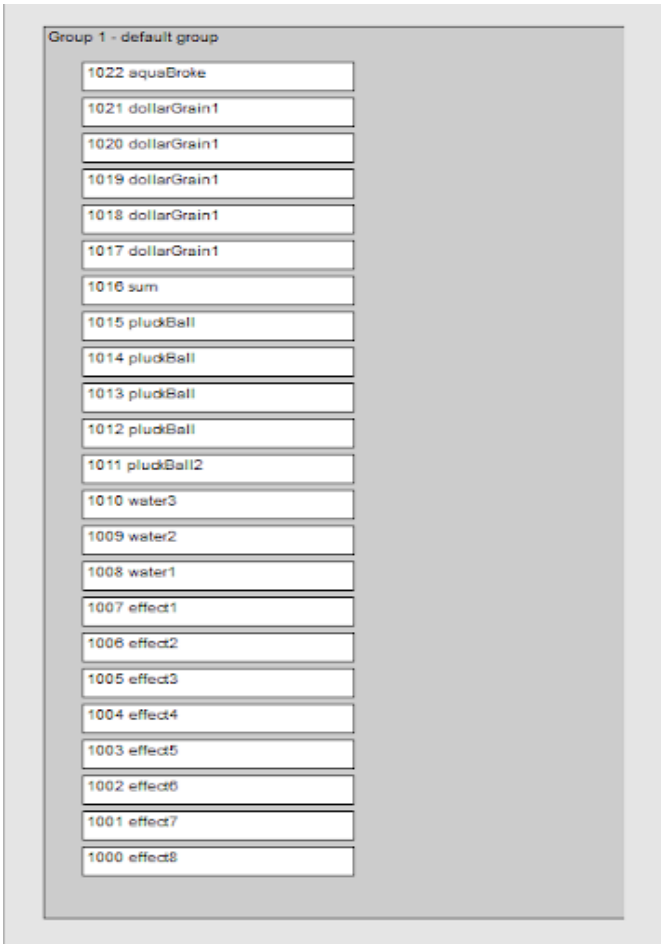
Zvuková realizace je zas rozdělena na několik částí a založená hlavně na směrování signálů. Prvně jsou primitivní SynthDefy, v tomto kroku však zatím nejsou definované vstupy a výstupy jako argumenty. Dvě funkce *In.ar()* a *Out.ar(),* které čtou a zapisují signál z a do sběrnic. Efektem jako takovým je shodou okolností UGen *BMoog()*, rezonantní filtr 4 řádu s měnitelným typem (dolní, horní a pásmová propust) a sklonem 24db/oct.

Další částí je směřování podle příchozích zpráv. Ty jsou ve tvaru nul a jedniček v poli, jednička - aktivní filtr, nula - neaktivní, indexace pole odpovídá indexaci filtru. Na začátku se definuje počítadlo, které se inkrementuje s každou jedničkou v poli a postupně každé instanci filtru přiděluje vstupy a výstupy. Využívá se pomocných uživatelských sběrnic 9 -12 a 57 až 72. Počet sběrnic se volí s ohledem na stereo signál a stejně tak i inkrementace počítadla. Routování ostatních nástrojů na vstup k filtrům se provádí předtím za použití jednoduché podmíněné funkce. Příchozí zprávu přiřadí na výstup instanci Synthu ostatních nástrojů. Zpráva přichází rovnou buď s hodnotou 0 - číslo sběrnice pro finální výstup; nebo 57 - číslo vstupu prvního filtru, počáteční hodnota počitadla. Dalším krokem je část kódu, ve které v

cyklu procházíme pole filtru. Příchozí routovací zprávu od konce ověřujeme a zjišťujeme, jestli je filtr aktivní. Ve chvíli, když narazíme na první aktivní, tak mu přidělíme na výstup 0, aby celá série filtrů nezůstala v uživatelských sběrnicích spolu se zvukem.

Poslední je cyklus, který vynuluje všichni výstupy pro všechny aplikace v případě, že v poli se nenajde ani jeden aktivní filtr.

Důležitým detailem u této aplikace, založené na směrování, je pořadí vyvolaných instanci SynthDefu - Synthu. Nestačí jenom poslat vše do patřičných sběrnic. Pro správný chod musí být zajištěno, že efektové Nody budou vyvolané až po těch, které generují zvuk. Případně budou zařazeny na konec Nodového stromu jako na Obr.18.

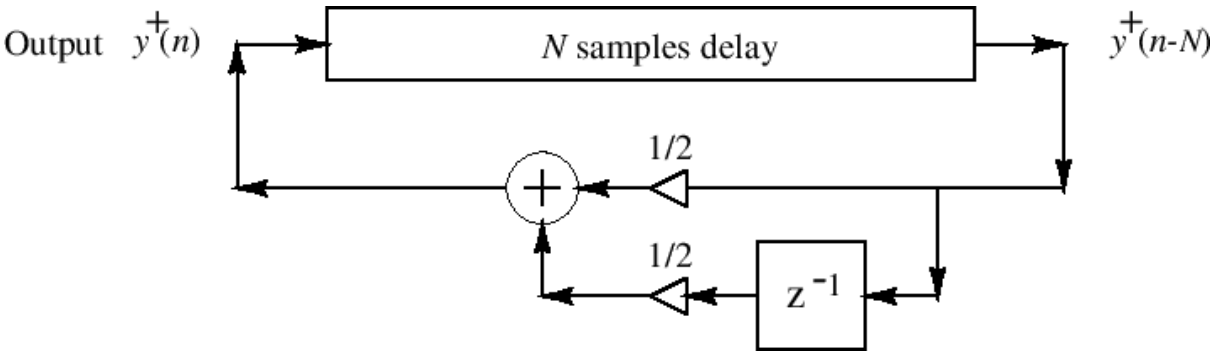


Obr. 18: Node tree se správným pořadím vyvolaných Synthu

9.4 BallsApp

Tato část programu byla reprezentována v mojí semestrální práci jako dílčí výsledek, byla kompletní a funkční, měla však procesní výsledky, které šlo zlepšit. Její obnovení, které jsem realizoval, nenahrazuje původní verzi, používá jiné metody, které mají svoje výhody a nevýhody.

Práce s generováním zvuku nám vždycky dá na výstupu čistější zvuk, než u přehrávání samplů, protože se vyhneme převodům a kvantizačnímu a dalším šumům. Ještě před rozhodnutím samply přehrávat jsem zkoušel syntetizovat zvuky materiálu, které se vyskytují v aplikaci, pomocí FM syntézy. Avšak nebylo to uspokojivé. Později jsem objevil modul Pluck(), UGen implementující Karplusuv-Stronguv algoritmus. (viz. Obr.19)



Obr. 19: Schema Karplusuva-Stronglova algoritmu [44]

Je zaměřen na fyzikální modelování strunných nástrojů a jeho princip spočívá v tom, že na vstup přivedeme excitační signál, v originální variantě a v našem případě bílý šum. Signál pošleme po zpožďovací lince do zpětné vazby skrz jednopólový filtr dolní propusti. Toto zapojení imituje reálnou oscilaci struny - zpožďovací linka, a její útlum - filtrace.[45]

Při manipulaci s časy zpoždění, útlumu a charakteristikou filtru jsem docílil potřebných zvuků, aby blízce imitovaly zvolené fyzické materiály. Ovšem nechybí tomu určitá míra variativnosti, kterou jsem navázal na parametry syntezátorů. Takto vytvořený systém je rozhodně stabilnější a víc pod kontrolou, než původní varianta se samply. Nejsou to reálné zvuky, ale výsledky subjektivně jsou dost přiblížené realitě, aby byla použita tato verze. Navíc během zpracování pomocí samplů nepoužíváme obrovskou knihovnu zvuků, ale jen několik, které modifikujeme. Touto modifikací se od reality jak objektivně, tak subjektivně taky hodně vzdálíme.

Výjimkou je zvuk metalické koule, nešlo ho napodobit pomocí modulu *Pluck()*, proto jsem vytvořil SynthDef navíc, sestavený ze sinusového oscilátoru, limiteru, filtru podobnému zmíněnému v aplikaci WaterApp filtru *Resonz* - tentokrát *Ringz* (jejich rozdíl spočívá v tom, že Ringz oproti Resonz má naopak proměnlivé zesílení a konstantní šířku pásma). Filtr zvýrazňuje 800Hz a v kombinaci s reverberací přidává metalické zabarvení sinusovému průběhu.

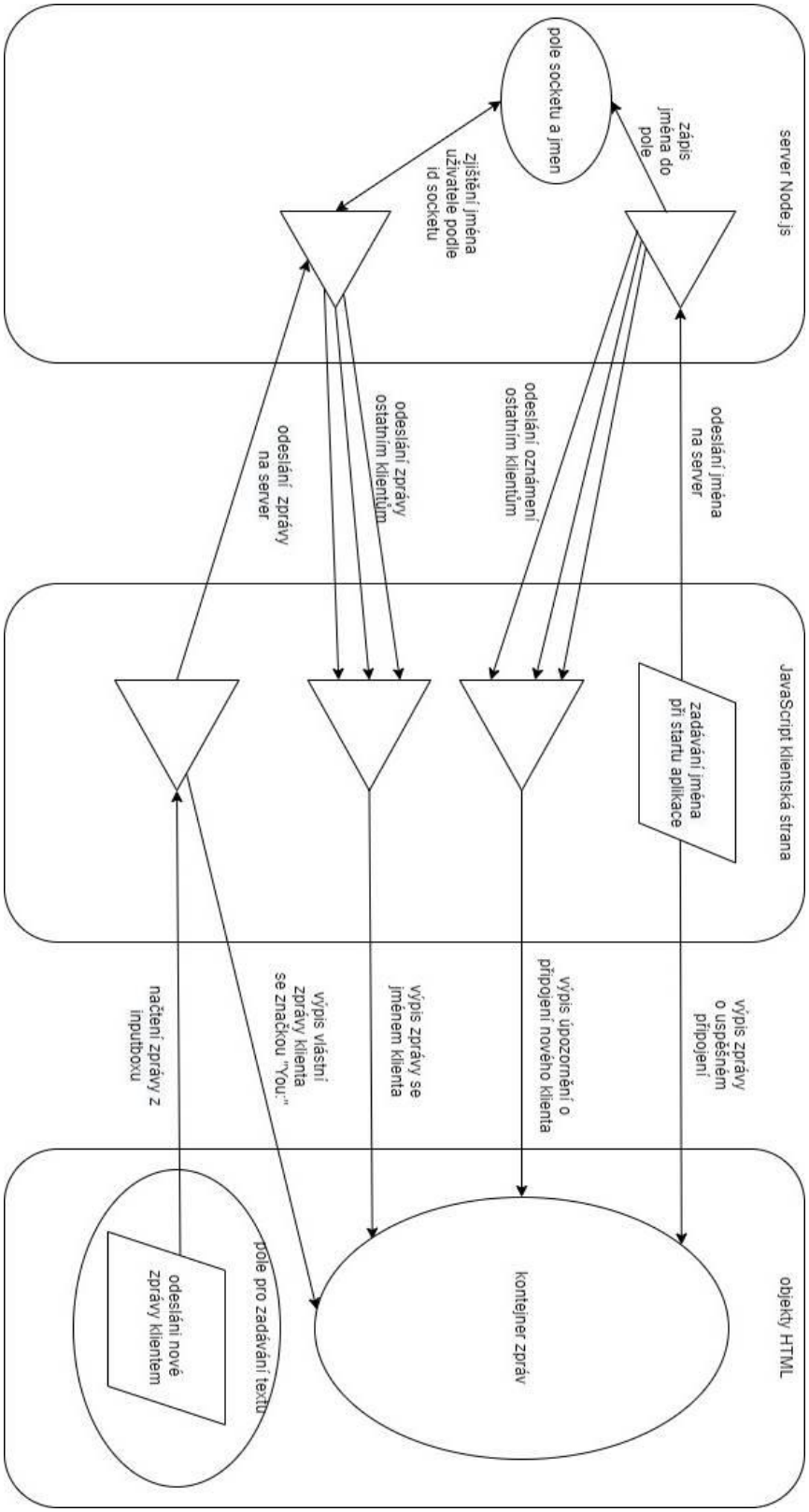
Nad tímto UGenem jsou potom jenom reverberace pro emulaci prostředí a panorámování pro rozpoložení ve stereu a orientaci posluchačů, kde dochází k nárazům.

Jedno z později přidaných omezení je opatření maximálního počtu kuliček. Na rozdíl od aplikace s bankovkou počet kuliček je uložen v proměnné na klientské straně a není potřeba nikam nic posílat. Při dosažení maximální hodnoty se zas vykreslí upozornění textem na okraji canvase. Tato podmínka, ověřující maximální počet kuliček, je přidána taky na funkce přidávání dalších kuliček po jedné a po několika.

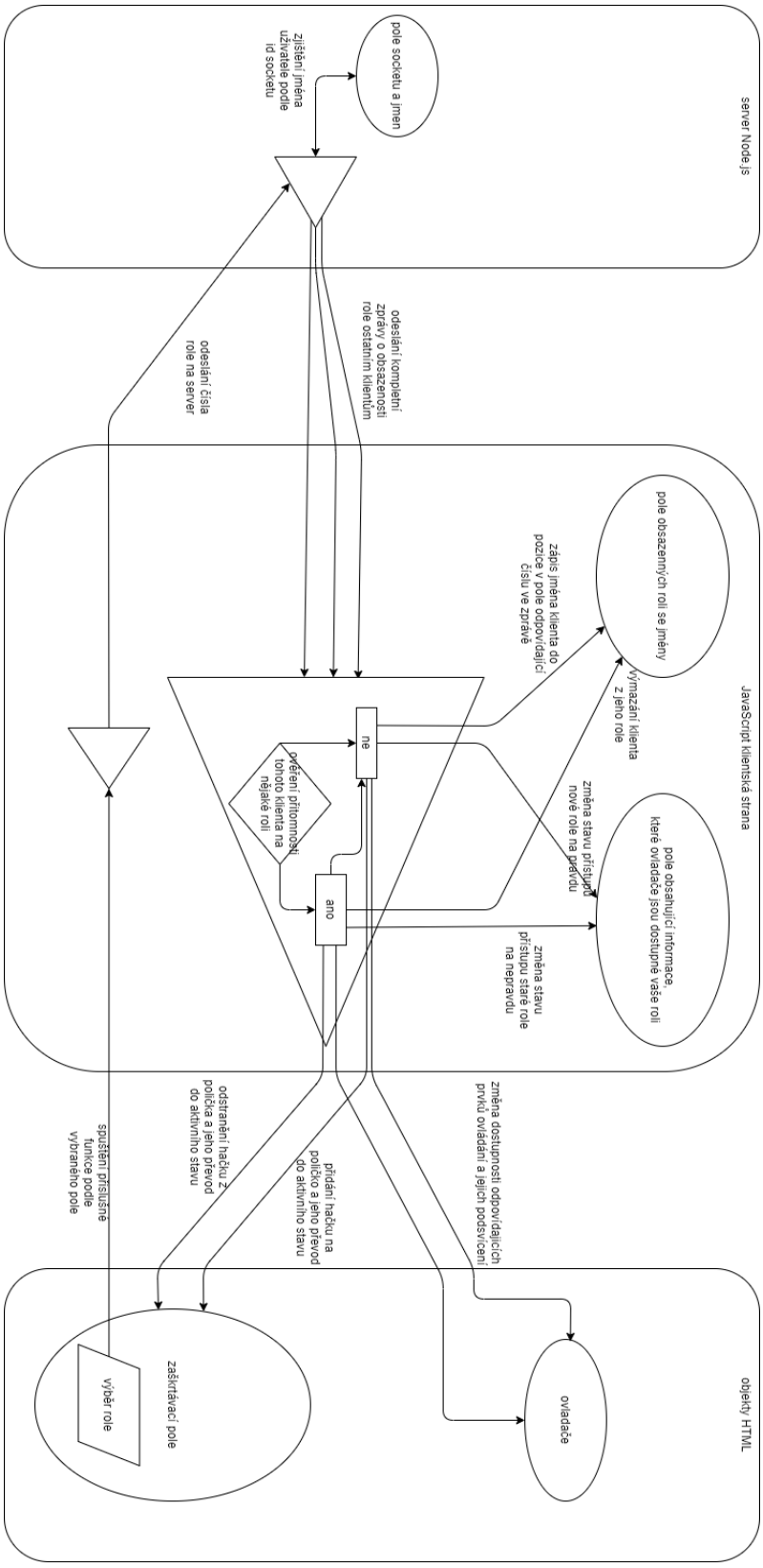
9.5 Rozdělení role a chat

Jedním z hlavních rysů aplikace je víceuživatelský přístup. Ke smysluplnému ovládání jednoho z nástrojů je potřeba rozdělit role a navrhnout nástroj pro jejich zvolení, zobrazení a návaznost na jednotlivé syntezátory. Tato funkcionalita zároveň souvisí s chatem a registraci. (viz. Obr.20)

Při otevření aplikace se nejprve objeví okno s nabídkou vyplnit svoje jméno. Toto jméno v budoucnu použijeme jako ověřování a identifikátor. Ukládáme ho do pole jmen na serveru. Celkově celý chat využívá web socketu a html elementů. Zásadní je jedna funkce, přidávající zprávy do kontejneru zpráv. Vytváří vždy nový element ‘div’, naplní ho textem, který dostane z inputboxu, přidá ho na konec kontejnerů a přejede (scroll) tam. Uživatel píše zprávu do inputboxu a při zmáčknutí potvrzovacího tlačítka se ten text přidá jako zpráva pro tohoto uživatele se známkou “You:”, odešle se na server, kde dostane podle id jméno a odtud se dostane na klientskou stranu jiných uživatelů a zobrazí se i pro ně, tentokrát se jménem toho uživatele. Podobně se zobrazují zprávy o připojení a odpojení uživatelů. Když se vrátíme zpátky k rozdělování rolí, tak použijeme zas podobný princip. Celý algoritmus je na Obr.21 a spočívá v následujícím: každé roli odpovídá zaškrtávací políčko s patřičným názvem, každý uživatel by měl zastupovat maximálně jednu roli najednou.



Obr. 20: Schema chatu



Obr. 21: Schéma procesu rozdělení role

Pro organizaci mechanismu rozdělení rolí jsem vytvořil 2 pole. První obsahuje jméno uživatele a při zaškrtnutí políčka se o tom pošle zpráva na server, vezme se jméno uživatele podle ID, pošle se zpátky klientovi. V dalším kroku se ověří, jestli v poli role existuje momentálně toto jméno, pokud ano, tak se vymaže a políčko s indexem uloženého jména se deaktivuje. Následně se připíše toto jméno příslušné zakliknuté pozici. Druhé pole obsahuje hodnoty pravdy a nepravdy podle toho, odpovídá-li jméno aktivního uživatele jménu uživatele na této pozici v poli jmen rolí. Při jakékoliv aktivitě v oblasti jednoho z nástrojů se nejprve ověří, jestli na to uživatel má právo podle tohoto pole, nebo ne. Následně se ta akce buď provede, nebo ne. Stejně tak, jak u odesílání zpráv server posílá oznámení o odpojení uživatele, odešle taky i příkaz uvolnit (vymazat z pole a deaktivovat políčko) role. Pro znepřístupnění role nebo neodepínání z role někoho jiného zaškrtnuté políčko se stane nedostupným až do chvíle, než se uvolní. Je promyšlená samozřejmě i role posluchače. Tato role nemá svoje políčko a posluchačem se stáváte automaticky po přihlášení. Poté, když jste vybrali nějakou roli, nemůžete se z ní odhlásit, protože i pro vás se to políčko stane nedostupným. Odhlásíte se a uvolníte místo tím, že si vyberete jinou roli, nebo zmáčknete tlačítko "stát se posluchačem".

Prvně jsem vytvořil slidery pomocí interních funkcí p5.js knihovny. Ty zjednodušují html gui a objekty. Potřeboval jsem pro rozdělení rolí někomu povolit hýbat s těmi slidery a někomu zakázat. Atribut disabled ovšem přepínal objekt slideru do nedostupného stavu jen jednou, a pak už to nešlo dynamicky měnit. Vedlo to k tomu, že jsem stejně tak jako u zaškrtávacích polí musel vyvolat v javascriptovém kódu objekty html a potom je zpracovávat. Aby byly dostupné v částech kódu, odpovídajících jednotlivým nástrojům, a zároveň byly zpracované společnou logikou, všechny slidery byly zavedeny jako globální proměnné. Spolu s deklarací a aktivací/deaktivací slideru do hlavní části a globálního přístupu byla přenesena i jejich komunikace t.j. odesílání a synchronizace aktuálních hodnot slideru ve chvíli, kdy jimi někdo potáhne. Při spuštění programu musí být definovány počáteční hodnoty sliderů. a zároveň musí být načteny na začátku kódu zvlášť. Jinak dojde k přečtení nedefinované hodnoty a program se nenastartuje.

Stejně tak pomocí html atributů jsem poskytl uživateli zobrazení těch prvků, které byly pro jeho roli aktivní či neaktivní. Jsou naprogramované dvě verze: v

jedné se skryjí všechny nedostupné prvky, což je pro uživatele srozumitelnější; v druhé verzi naopak se zvýrazní barevně ovladače dostupné, což funguje líp z hlediska designu.

Obsazení uživatelem jedné z rolí vede k její nedostupnosti pro ostatní. Když je tento uživatel neaktivní, nezavřel aplikaci a tak dále, vzniká problém. Opatřil jsem ho funkcí activityWatcher(), ve které jsou definované: čas uplynulý od poslední aktivity, maximální čas bez aktivity a seznam akce, které se počítají za aktivitu (pohyb, klik nebo scrollování myší, zmáčknutí tlačítka klávesnice). Každou vteřinu čas uplynulý od poslední aktivity se zvyšuje o jednu sekundu a při detekci jedné z aktivit se vynuluje. Pokud tento čas přesáhne maximální, uživatel se odhlásí z role a přejde do stavu posluchače a zároveň se mu zobrazí okénko s oznámením o jeho neaktivitě. Běh aplikace se pro něj zastaví až do chvíle, dokud to nepotvrdí přečtením upozornění (zmáčknutím tlačítka “OK” na otevřeném okně zprávy).

9.6 Instance mode a celek programu

Až na zmíněné globální objekty GUI a jejich komunikaci a logiku každý z nástrojů je nezávislý sketchový soubor a samostatný program. Potřeboval jsem je shromáždit do jednoho. K tomu p5.js poskytuje tzv. Instance mode. Pomocí něj z každého sketche můžeme vytvořit objekt nebo šablonu a potom ji vyvolávat třeba i několikrát, nebo více různých instancí v jednom souboru. Součástí instance je namespace - prakticky je to jméno, pod kterým se shromáždí veškeré funkce a proměnné konkrétní šablony. Přes toto jméno se přistupuje ke sketchy zvenku.

Když máte jeden p5.js nástroj ve tvaru jednoho souboru-sketche, tak socket pro komunikaci je samozřejmě definován uvnitř tohoto souboru. Když více nástrojů spojíte v jednom souboru a každý bude mít svoji instanci, tak jejich sockety vám vytvoří víc spojení - „uživatelů“. Ty jsem samozřejmě taky přenesl a sjednotil do jednoho ve společném globálním prostoru, spolu se socketem pro chat.

Pro pozadí stránek jsem vytvořil v globální *setup()* funkci canvas. Abych ho posunul dozadu, použil jsem html atribut "z-index", zodpovídající za vrstveni a pořadí zobrazení stránek.

Rozpoložení prvků html jsem naprogramoval pomocí CSS. Většinou jsem použil fixní a absolutní hodnoty. Rozpoložení každého canvase je taky absolutní a je uvedené (uvnitř p5,js sketche jako vlastnost objektu) ve tvaru souřadnic v pixelech s počátkem (0,0) v levém horním rohu. Nejzajímavějším detailem je scrollování chatů, které se objeví při přeplnění kontejneru zprávami.

V tomto globálním setupu jsou kromě všeho vytvořeny dva obdélníky. Jeden slouží jako estetický podkres pro chat pro počáteční fázi, kdy ještě není vyplněn zprávami. Druhý je rozložen v pravé části obrazovky a na něm se promítají předem nadefinované texty v závislosti na poloze myší. Slouží jako komentář, popis nebo momentální help soubor.

9.7 Rozšíření

V SuperCollideru existuje několik rozšíření, mezi nimi je většina uživatelských, které nejsou moc spolehlivé. V řadě s nimi je taky dost oficiálně ověřených a zároveň v porovnání se základními UGeny pokročilejšími. Nejpopulárnějšími a doporučenými jsou sc3-pluginy.[46]

Dalším rozšířením, kterého jsem využil v rámci node.js při rozvoji aplikaci, je nodemon. Je to služba, sledující změny ve vašem zdrojovém kódu a při jejich registraci restartuje server. Instaluje se balikem v npm.

10. OPTIMALIZACE, RIZIKA A OMEZENÍ

V rámci SuperCollideru jsou UGeny jedním z nejnáročnějších prvků na CPU (mimo některých dynamických GUI prvků). GUI však má při real-time realizaci až sekundární prioritu, proto pokud chybí výkon výpočtu audia, nepočítá se grafika. Pokud ale nastane situace, že i přesto procesor nestíhá, zvuk začne přeskakovat a vznikají další nechtěné artefakty.

Jedním ze způsobů jak minimalizovat výpočetní náročnost UGenu je používat místo audio ratu (.ar) control rate (.kr). Při real-time zpracování toto řešení může značně pomoci, při potřebě kvalitnějšího renderu potom stačí nastavit parametr serveru blocksize na hodnotu 1 a všechny rate se změní zpět na audio (což znamená, že budou eliminovány interpolační šumy vzniklé vlivem control rate).

Dalším způsobem je regulovat množství routovaných signálů a zbytečně neplýtvat výkonem provozu nevyužitých bloků. Například v situaci, kdy jsou signály reprezentované jedním SynthDefem a následně rozmnoženy vyvoláním několika Nodů (několik instancí) tohoto SynthDefu, je efektivnějším řešením rozmnožit signál už ve stadiu generace signálu. Například je možno použít jednoduchý generátor SinOsc a potom aplikovat processing (obálku, filtry a panorámování) na stejný signál víckrát - efektivně jsme tedy ušetřili n UGenů a nahradili jej rozdělením na n signálů. [6]

Podobným způsobem je třeba uvažovat i při složitějším routování a při organizaci Node Tree ve vrstvě Nodu a Group.

Určité úrovně optimalizace můžeme dosáhnout, když přenecháme část výpočtů na klientské straně místo serverové. Jde to udělat u některých procesů jako jsou například konvertace dB úrovně na lineární a opačná.

Některé Ugeny jako zpožďovací linky nebo filtry používají interpolaci a v argumentech těchto funkcí můžeme volit různé její typy nebo dokonce její absenci. V případech dynamického zpracování samozřejmě stojí za to použít kubickou interpolaci nebo některý její jiný typ, který dá kvalitnější výsledek. Ovšem když máme například fixní hodnoty zpoždění, má smysl si otestovat, jestli jednodušší typ

interpolací jako jsou lineární nebo neinterpolovaný signál vůbec budou znít dostatečně kvalitně, bude- li ten rozdíl vůbec poznat. [23]

Pro přehlednost těchto směrovacích procesů je doporučené a logické taky použití sběrnic. Princip zůstává stejný, při vyvolání několika UGenu nebo Nodu dohlížíme na to, jestli nejde nějakou jednotku od skupiny oddělit a použít jenom jednu její instanci místo několikerého klonování. [6]

Dalšími faktory, které mohou zatížit server nebo kriticky porušit zážitek ostatních uživatelů, jsou hackeři a lidský faktor. Opatření, které tomuto chování zamezí, jsou:

- vhodné omezené rozsahy parametrů (tady je nepřecenitelně důležité stanovit hodnoty velmi opatrně, aby tyto rozsahy nebyly malé na experimentální tvorbu a nebyly rušící pro ostatní. Například šumová struktura s ohledem na existenci noise hudby jako žánru ještě není důvodem k omezení, ale je jím_vznik zpětné vazby, překročení nějaké určité hodnoty střední energie ve zvuku.)
- individuální mix panel (původně byla zamýšlená koncepce zvláštní role pro mixéra nebo zvukaře, ale není to kontrolovatelný process a mohlo by to nepříjemně narušit zkušenost ostatních hráčů, proto podobně jako v projektu Web Média Sequencer bylo rozhodnuto o zvlášť mixu pro každého uživatele)
- počet místností a přihlášených uživatelů. Nejzákladnější problém, jehož řešení je správné nastavení parametrů serveru metodami jako jsou *.maxNodes = value* a další.
- hackerské útoky typů XSS anebo cross-site scripting nemají k útokům mnoho cílů, protože k tomu používají dynamické DOM nebo vstup. Ani jeden z_těchto prvků v rámci programu zatím nebyl použit. V případě použití vstupu, například pro zadávání jména, se musí odfiltrovat "nebezpečné" znaky: < za <, > za > atd [37]
- hackerským útokům typů DoS nebo DDoS zde nehodlám zabraňovat, protože rozsah aplikace to nevyžaduje. V budoucnu podle potřeby se mohou použít za tímto účelem SYN-cookies

Některé UGeny jako například zpoždění a reverberace potřebují hodně “real-time paměti”, protože ukládají část signálu k pozdějšímu zpracování nebo přehrávání. Ovšem ne do klasického zásobníku, ale do interní paměti, která je alokovaná na serveru. V případě, že takových UGenu je více, je pravděpodobné, že se vyskytne chyba: 'exception in real time: alloc failed'. Ošetřil jsem tuto chybu rozšířením paměti výchozího nastavení z 8192 kilobytů příkazem: *s.options.memSize* na 65536 kilobytů.

11. ZÁVĚR

11.1 Hodnocení výsledků

V rámci diplomové práce byla provedena rešerše existujících řešení, problematiky interaktivity a rozhraní. Byly popsány technologické postupy, sestaven model a navržena aplikace. Program byl realizován celý s čtyřmi zvukovými nástroji, chatem a popisujícím informačním blokem, ověřen kompletní model komunikace a provedena optimizace a zamezení rizikům.

Koncept s několika místnostmi nebyl realizovan v rámci práce. Jeho realizace je celkem jednoduchá z hlediska programování, už vybrané a použité nástroje samy o sobě jsou k tomu postačující. Ovšem každá místnost znamená server navíc, kvůli vysílání dalšího zvukového toku.

Časově nejnáročnější částí práce bylo zkoumání jednotlivých zvukových modulů a jejich kombinace, zjišťování a hledání spoluznějících tembrálních charakteristik a vhodných rozsahů a mezních hodnot. A taky občas obcházení nedokonalosti a nedostatečně rozvinutému procesu debuggingu obou prostředí - SuperCollideru a p5.js.

Rozměry vizuálních ploch-canvasu jsou fixní a některé závisí na obrázcích na pozadí. Kvuli rozpoložení na jedné stránce a kvli tomu, že nejsou přizpůsobeny rozměrům obrazovky, je aplikace zatím určena jen pro PC a větší obrazovky. Existuje možnost dynamicky měnit rozměr canvasu, ale i přes moji snahu nechat všechno v relativních hodnotách, dost elementů, zásadních pro aplikaci, operuje s konkrétními čísly pixelů a rozměry, což vede k zamítnuti této možnosti.

11.2 Plány na rozvoj aplikace

Přesto, že aplikace v provozu nevykazuje žádné zásadní problémy, prošla několika úpravami na odstranění chyb. Přesto existuje prostor pro vylepšení, který bych mohl vyplnit následujícími návrhy:

- Implementovat výběr bankovek u nástroje DollarApp a asociovat každou jinou bankovku s jinou skladbou autora příslušného státu, kterému bankovka patří.
- Implementovat výběr obálky pro granule u nástroje DollarApp
- V nástroji FilterApp implementovat jako zdroj zvuku živý vstup z mikrofonu.

LITERATURA

[1] Sonoristics or sonoristic technique [online] [cit. 2019-12-18]. Dostupné z: http://musicinmovement.eu/glossary/sonoristics-sonorism

[2] Sonoristics, sonorism. Grove Music Online. October 22, 2008. Oxford University Press. Date of access 18 Dec. 2019 [online] [cit. 2019-12-18]. Dostupné z: https://www.oxfordmusiconline.com/grovemusic/view/10.1093/gmo/9781561592630.001.0001/omo-9781561592630-e-0002061689

[3] THOMAS, Adrian. *Polish Music since Szymanowski* [online]. New York: Cambridge University Press, 2009 [cit. 2019-12-18]. DOI: 10.1017/CBO9780511482038. ISBN 9780521582841.

[4] ZBIGNIEW, Granat. *Rediscovering 'Sonoristics': A Groundbreaking Theory from the Margins of Musicology*. 2009. In: Music's Intellectual History, edited by Zdravko Blažeković and Barbara Dobbs Mackenzie. New York: Répertoire International de Littérature Musicale, 821–833.

[5] *The Primer Spectral Composition* - The Wire, Issue 237, 09.03

[6] COTTLE, David Michael. *Computer Music with examples in SuperCollider 3*. August 2005.

[7] GUREVICH, Michael. *JamSpace: Designing A Collaborative Networked Music Space for Novices. NIME* (2006).

[8] MILETTO, Evandro Manara, Marcelo Soares PIMENTA, François BOUCHET, Jean-Paul SANSONNET a Damián KELLER. *Principles for Music Creation by Novices in Networked Music Environments.* Journal of New Music Research [online]. 2011, 40(3), 205-216 [cit. 2019-12-18]. DOI: 10.1080/09298215.2011.603832. ISSN 0929-8215. Dostupné z: http://www.tandfonline.com/doi/abs/10.1080/09298215.2011.603832

[9] SCHNELL, Norbert, Sébastien ROBASZKIEWICZ. *Soundworks – A playground for artists and developers to create collaborative mobile web performances.* `Proceedings of the Web Audio Conference (WAC'15), 2015, Paris, France. ⟨hal-01580797⟩

[10] Contexts of Collaborative Musical Experiences Tina Blaine Entertainment Technology Center Carnegie Mellon University Pittsburgh, Sidney Fels University of British Columbia Dept. of Electrical & Computer Engineering. Proceedings of the 2003 Conference on New Interfaces for Musical Expression (NIME-03), Montreal, Canada

[11] Ableton Link Documentation [online] [cit. 2019-12-18]. Dostupné z: https://ableton.github.io/link/

[12] Avid Knowledge Base [online] [cit. 2019-12-18]. Dostupné z: http://avid.force.com/pkb/articles/en_US/FAQ/What-can-I-do-with-Avid-Cloud-Collaboration-for-Pro-Tools?retURL=%2Fpkb%2Farticles%2Fen_US%2Ffaq%2FPro-Tools-Cloud-Collaboration-FAQ&popup=true

[13] NINJAM, cockos incorporated [online] [cit. 2019-12-18]. Dostupné z: https://www.cockos.com/ninjam/

[14] GEIGER, Gunter. *The reacTable* - A new table based computer music instrument.* Music Technology Group IUA-UPF Barcelona October 1, 2004

[15] FAVORY, Xavier; SERRA, Xavier. Multi Web Audio Sequencer: Collaborative Music Making. *arXiv preprint arXiv:1905.06717*, 2019.

[16] Soundworks v.2.2.5 [online] [cit. 2019-12-18]. Dostupné z: http://collective-soundworks.github.io/soundworks/

[17] MARINI, Joe. *The Document object model: processing structured documents*. 2. New York: McGraw-Hill/Osborne, c2002. ISBN 0072224363.

[18] CROCKFORD, Douglas. *JavaScript: the good parts*. Sebastopol: O´Reilly, 2008. ISBN 9780596517748.

[19] ŽÁRA, Ondřej. *JavaScript: programátorské techniky a webové technologie*. Brno: Computer Press, 2015. ISBN 978–80–251–4573–9.

[20] ROADS, Curtis. *The computer music tutorial*. Cambridge, Mass.: MIT Press, c1996. ISBN 9780262181587.

[21] BROWN, Ethan. *Web development with Node and Express*. Sebastopol, CA: O'Reilly, 2014.

[22] HERMANN, Thomas, Andy HUNT a John G. NEUHOFF. *The sonification handbook*. Berlin: Logos Verlag, 2011. ISBN 978-3-8325-2819-5.

[23] WILSON, Scott, Nick COLLINS a David COTTLE. *The SuperCollider book*. Cambridge, Mass.: MIT Press, c2011. ISBN 9780262232692.

[24] Client vs Server. SuperCollider Help [online] [cit. 2019-12-18]. Dostupné z: https://depts.washington.edu/dxscdoc/Help/Guides/ClientVsServer.html

[25] *Learn Javascript with p5.js: coding for visual learners*. New York, NY: Springer Science+Business Media, 2018. ISBN 978-1-4842-3425-9.

[26] YOSPANYA, Poonna, BHATIA, Sanjiv K., Shailesh TIWARI, Krishn K. MISHRA a Munesh C. TRIVEDI. *Remote Collaborative Live Coding in SuperCollider-Based Environments via Open Sound Control Proxy*. ed. Advances in Computer Communication and Computational Sciences [online]. Singapore: Springer Singapore, 2019, 2019-05-22, s. 47-54 [cit. 2019-12-18]. Advances in Intelligent Systems and Computing. DOI: 10.1007/978-981-13-6861-5_4. ISBN 978-981-13-6860-8. Dostupné z: http://link.springer.com/10.1007/978-981-13-6861-5_4

[27] REAS, Casey a Ben FRY. *Processing: a programming handbook for visual designers and artists*. Second edition. Cambridge, Massachusetts: The MIT Press, [2014]. ISBN 9780262028288.

[28] CHOPRA, Varun. *Essential WebSockets – Building Apps with HTML5 WebSockets*. 2015. Packt Publishing

[29] WRIGHT, MATTHEW. *Open Sound Control: an enabling technology for musical networking. Organised Sound* [online]. 2005, **10**(3), 193-200 [cit. 2019-12-18]. DOI: 10.1017/S1355771805000932. ISSN 1355-7718. Dostupné z: https://www.cambridge.org/core/product/identifier/S1355771805000932/type/journal_article

[30] WRIGHT, Matthew, Adrian Freed, Ali Momeni. *OpenSound Control: State of the Art 2003*. Proceedings of the 2003 Conference on New Interfaces for Musical Expression (NIME-03), Montreal, Canada NIME03-153

[31] De Man, Brecht & Reiss, Joshua. *A knowledge-engineered autonomous mixing system.* 2013

[32] Jak funguje syntezátor. POPIS JEHO HLAVNÍCH ČÁSTÍ 29/03/2013 [online] [cit. 2019-12-18]. Dostupné z: http://elektronicka-hudba.telotone.cz/clanky/syntezator

[33] An intoduction to filters | Techhnical Articles [online] [cit. 2019-12-18]. Dostupné z: https://www.allaboutcircuits.com/technical-articles/an-introduction-to-filters/

[34] Jan Robenek, *Bleskový návrh kmitočtových filtrů.* 14. Leden 2009 [online] [cit. 2019-12-18]. Dostupné z: https://vyvoj.hw.cz/teorie-a-praxe/bleskovy-navrh-kmitoctovych-filtru.html

[35] Pan2 | SuperCollider 3.10.3 [online] [cit. 2019-12-18]. Dostupné z: http://doc.sccode.org/Classes/Pan2.html

[36] White Noise Definition vs Pink Noise [online] [cit. 2019-12-18]. Dostupné z: https://www.acousticfields.com/white-noise-definition-vs-pink-noise/

[37] Cross-site scripting [online] [cit. 2019-12-18]. Dostupné z: https://cs.wikipedia.org/wiki/Cross-site_scripting

[38] GALLOWAY, Jon, Brad WILSON, K. Scott ALLEN a David MATSON. *Professional ASP.NET MVC 5*. Indianapolis, IN: Wrox, a Wiley brand, [2014]. Wrox professional guides. ISBN 978-1-118-79475-3.

[39] HERRON, David. *Node Web Development.* Packt Publishing, August 2011

[40] Perlin Noise [online] [cit. 2020-05-07]. Dostupné z: https://flafla2.github.io/2014/08/09/perlinnoise.html

[41] SEBASTIEN ST-LAURENT. *Shaders for Game Programmers and Artists.* Thompson Course Technology, 2004.

[42] Free STEMS from Cambridge University [online] [cit. 2020-05-07]. Dostupné z: http://cambridge-mt.com/rs/bkg

[43] Syntezator Animoog [online] [cit. 2020-05-07]. Dostupné z: https://www.moogmusic.com/products/animoog

[44] Karplus-Strong Algorithm. Stanford University [online] [cit. 2020-05-07]. Dostupné z: https://ccrma.stanford.edu/~jos/pasp/Karplus_Strong_Algorithm.html

[45] Karplus-Strong Algorithm scheme. Stanford University [online] [cit. 2020-05-07]. Dostupné z: https://crypto.stanford.edu/~blynn/sound/karplusstrong.html

[46] SC3 plugins [online] [cit. 2020-05-07]. Dostupné z: https://supercollider.github.io/sc3-plugins/

[47] Perlin Noise [online] [cit. 2020-05-07]. Dostupné z: https://www.khanacademy.org/computing/computer-programming/programming-natural-simulations/programming-noise/a/perlin-noise

[48] K. STEIGLITZ, *A Note on Constant-Gain Digital Resonators*, Computer Music Journal, vol 18, no. 4, pp. 8-10, Winter 1994.

[49] Oppenheim, Alan V., Ronald W. Schafer, and John R. Buck. *Discrete-Time Signal Processing*. Upper Saddle River, NJ: Prentice Hall, 1999